Доля П.Г.
Харьковский национальный университет им. В.Н. Каразина
факультет математики и информатики
кафедра теоретической и прикладной информатики
2015 г.

Mathematica для математиков.

Когда-то телескоп позволил людям увидеть дальше, а микроскоп ближе. *Mathematica* - инструмент сегодняшнего дня. Она позволяет увидеть дальше и ближе в математике и обнаружить идеи, которые являются новыми для нас, а иногда и для всего мира.

Из книги Theodore W. Gray and Jerry Glynn. The Beginner's Guide to Mathematica. Version 2. (Addison-Wesley, 1992).

Система символьных вычислений *Mathematica* очень обширна и многогранна. Она содержит почти 4000 стандартных функций и необозримое число функций в пакетах расширения. В настоящем пособии мы даем описание только некоторых функций, которыми, как нам кажется, должен владеть каждый математик. Пособие предназначено в первую очередь для знакомства с математическими возможностями системы. Читайте текст и выполняйте примеры. Во многих случаях все пояснения дает само выполнение примера. Надеемся, что по окончании выполнения последнего примера вы начнете применять систему *Mathematica* для решения ваших задач. Уверяем, что в некоторых случаях вы будете удивляться получаемым решениям, но ваш математический кругозор после этого только расширится.

Примеры, приводимые в данном пособии, проверялись в версии пакета *Mathematica* 9.0. В более старых версиях часть примеров работать не будет. Особенности некоторых функций *Mathematica* 5 мы иногда описываем отдельно.

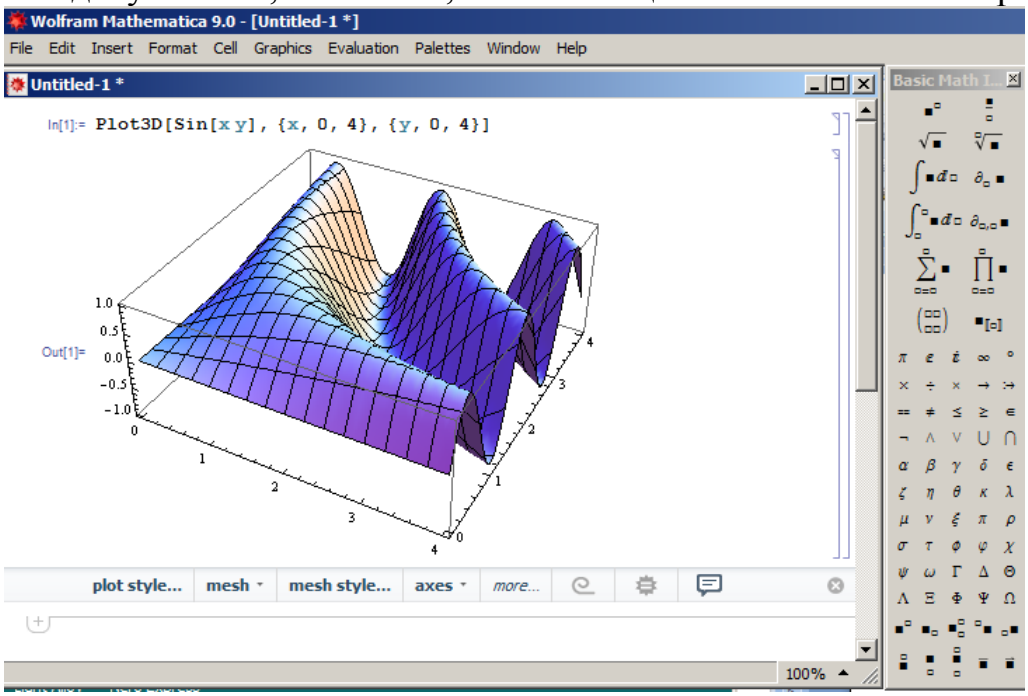
Большая часть, изложенного в пособии материала, доступна студентам младших курсов математических факультетов университетов, а также студентам технических вузов, прослушавшим курс высшей математики. Сложности могут возникнуть только при чтении параграфов, содержащих приложения системы *Mathematica* к решению прикладных задач, поскольку они предполагают некоторое знакомство с соответствующими областями знаний.

Весь материал многократно использовался в спецкурсе «Основы математических вычислений в системе *Mathematica*», читаемого автором студентам прикладного отделения механико-математического факультета Харьковского национального университета им. В.Н. Каразина, которые специализируются по прикладной геометрии. Некоторые части этого пособия мы излагали в курсе информатики для студентов других отделений механико-математического факультета.

Оглавление

Первые шаги

После запуска пакета Mathematica вы увидите стандартное меню программы, чистое окно документа и, возможно, панель специальных символов справа.



Внешний вид окна программы

Рис. 1.1

Наберите в окне документа: **2+2**. Теперь используйте комбинацию «горячих клавиш» **Shift-Enter** для того, чтобы Mathematica начала вычисления. Вы увидите:

In[1]:= **2 + 2**
Out[1]:=**4**

Справа от окна будут квадратные скобки, идентифицирующие разделы документа (секции) Отнеситесь к ним как к разметке различных листов бумаги, на которых вы делаете математические выкладки. Если вы хотите выполнить вычисление данной секции, курсор должен находиться в ее пределах. Символы стоящие слева In[1]:= и Out[1]:=, обозначают и нумеруют входные и выходные секции документа. Их наличие или отсутствие зависит от настроек программы.

В следующих разделах текст, набранный шрифтом **Arial-жирный** или стандартным формульным шрифтом системы *Mathematica* (стиль Input), вы можете вводить в окне документа и выполнять путем нажатия комбинации клавиш `Shift-Enter`.

Для математических операций в выражениях используются следующие знаки:

- ^ – обозначает возведение в степень; **2^3** это 2 в третьей степени.
- / – это деление; **34/89** это 34 делить на 89.
- * – это умножение; **34*89** это 34 умножить на 89.
- Пробел может служить для обозначения умножения: **34 89** это тоже, что и **34*89**. Однако пробелы часто ничего не обозначают; **Sin[x]** и **Sin [x]** это тоже что и **Sin[x].**
- ( ) – (круглые скобки) служат для указания порядка вычислений: **(x+3)/x** это $x+3$ деленное на x. Не используйте квадратные или фигурные скобки для указания порядка вычислений.
- [ ] – (квадратные скобки) используются для указания аргументов функций; **Sin[x]** - это синус от x. Если написать Sin(x), то это не будет работать.
- { } – (фигурные скобки) обозначают список; **{1,2,3,4}** - это список из четырех чисел.
- % представляет результат последнего вычисления. Например

In[2]:= **3*7**
Out[2]:= **21**
In[3]:= **%^2**
Out[3]:= **441**

- **!** обозначает факториал; **5!** это 5 факториал.
- = обозначает присваивание; ***a*=5** это a присвоить 5. Значения переменных, присвоенные или вычисленные в одной секции, доступны в других секциях.
- **:=** отложенное присваивание.
- == проверка на равенство; ***a*==5** возвращает True, если a равно 5.
- != проверка на неравенство; ***a*!= 5** возвращает True, если a не равно 5. Следите за пробелами в этом выражении: ***a*!=5** может обозначать a факториал равно 5 или a не равно 5.
- <,<=,>,>= обозначают неравенства.

Набранная формула, как правило, не заканчивается никаким символом. В этом случае результат вычисления будет отображаться в окне документа. Если вы желаете выполнить вычисления, а результат не выводить в документ, то завершайте такие формулы точкой с запятой.

Основой пакета являются символьные вычисления. *Mathematica* умеет выполнять алгебраические преобразования, решать уравнения и системы уравнений, вычислять интегралы в аналитическом виде, решать дифференциальные уравнения, и умеет делать многие другие виды символьных вычислений, но и численные вычисления в ней выполняются великолепно.

Mathematica может выполнять арифметические операции не только как калькулятор. Она может выполнить алгебраическую операцию, где ответ не просто число.

In[4]:= $\mathbf{27x + 8x}$
Out[4]:= 35 x

Она может разложить выражение

In[5]:= $\mathbf{Expand}[(\boldsymbol{a}+\boldsymbol{b})\text{^}\mathbf{3}]$
Out[5]:= $343 + 147b + 21b^2 + b^3$

или

Expand[(**1** − ***x***)(**1** + ***x*** + ***x*^2** + ***x*^3** + ***x*^4** + ***x*^5**)]

$1 - x^6$

Дальше мы не будем приводить идентификаторы In и Out входных и выходных секций документа. Кроме того, при переносе некоторых формул и результатов их вычисления из документа *Mathematica* в текстовый редактор, мы будем использовать команду копирования Copy As... – MathML (выделяем формулу, щелкаем правой кнопкой мыши и в выпадающем меню выбираем Copy As... – MathML). Затем вставляем формулу в документ. В текстовом редакторе, в котором мы создаем наше пособие, такое копирование создает формулы в виде близком к привычной математической форме.

Мы можем проинтегрировать выражение по переменной *x*:

$\int (\boldsymbol{a} + \boldsymbol{x})^{\mathbf{5}}\, \mathbb{d}\boldsymbol{x}$

Для набора формул в таком виде мы используем трафареты панели специальных символов Basic Math Input. Если панели нет на экране, то включить ее можно из меню: *Palettes – Other – Basic Math Input*. Результат последней команды будет следующим

$\frac{1}{6}(a + x)^6$

Символьные вычисления также имеют отношение к выражениям, в которые входят одни числа. Например, *Mathematica* может проводить вычисления с дробями и, приводя их к общему знаменателю, получать точный ответ.

$\frac{\mathbf{1}}{\mathbf{3}} + \frac{\mathbf{2}}{\mathbf{5}}$

$\frac{11}{15}$

Мы можем работать с числами большими настолько, что большинство калькуляторов их получить не могут, например, можно точно вычислить **200!**.

Основной принцип *Mathematica* – не делать никаких приближений кроме затребованных. Так выражение

$\sqrt{\mathbf{12}}$

будет преобразовано к виду

$2\sqrt{3}$

Mathematica преобразует выражение, но оставляет в записи ответа квадратный корень, не выполняя никаких округлений. Если мы хотим получить численное приближение, то можем использовать функцию N.

N[$\sqrt{\mathbf{12}}$]

3.4641

Функция *N*[…] может иметь два аргумента. Вторым необязательным аргументом можно определить точность результата (количество цифр)

N[$\sqrt{\mathbf{12}}$, **40**]

```
3.464101615137754587054892683011744733886
```

N[**π**, **40**]

```
3.141592653589793238462643383279502884197
```

Однако, если хотя бы одно из чисел задано с десятичной точкой, то и результат будет приближенным

$\sqrt{12.}$

```
3.4641
```

Если исходное число задано с точностью, превосходящей точность вычисления процессора (на персональных компьютерах с 32 – х битовым процессором это примерно 15 значащих цифр), то и результат будет иметь точность исходных данных. Сравните

$\sqrt{12.000000}$

```
3.4641
```

$\sqrt{12.0000000000000000}$

```
3.4641016151377546
```

Все встроенные в Mathematica имена начинаются с заглавной буквы (**Sin, Table, Factor** и т.д.), некоторые имеют несколько заглавных букв **ContourPlot**. Вы должны писать эти имена так, как показано или код не будет работать.

Аргументы функций заключают в квадратные скобки, а не круглые (**Sin[x], Factor[x^2-9]** и т.д.). Если вы хотите получить правильный ответ, не используйте **Sin(x)** или **sin(x)** или **sin[x]**, вы должны использовать только **Sin[x].**

Многие специальные символы, для которых нет соответствующих кнопок на клавиатуре, можно ввести, используя панель специальных символов, которая обычно расположена справа от окна документа (см. рисунок 1.1). Она позволяет вводить математические формулы в виде близком к естественной математической записи. Имеются и другие панели. Вывести панель на экран можно из меню *Palettes,* в котором следует выбрать имя – название одной из панелей. Панель специальных символов, показанная на рис. 1.1, имеет имя *BasicMathInput* и вызывается из меню *Palettes – Other – Basic Math Input.*

На панели спецсимволов (имя панели – *BasicMathInput*) имеются знаки интеграла $\int$, ряда $\sum$, корня $\sqrt{\ }$ и многие другие. При выборе мышью одного из них в окне документа появляется трафарет ввода, который надо заполнить обычными символами

$\int \mathbf{Sin}[x]\, dx$

$-\mathrm{Cos}[x]$

Однако возможен ввод математических знаков с помощью специальных имен функций. Последний пример можно выполнить так:

$\mathbf{Integrate}[\mathbf{Sin}[x], x]$

Ядро пакета *Mathematica* понимает только такие имена функций, а оболочка переводит специальные обозначения, например $\int \mathrm{Sin}[x]\mathrm{dx}$, в понятный ядру текст Integrate[Sin[x],x].

Вы можете набирать математические формулы либо в привычном математическом виде, используя спецсимволы, либо в текстовом формате, как это было показано в предыдущем примере для интеграла. Для набора большинства специальных символов есть особые комбинации клавиш.

Mathematica умеет работать с комплексными числами. В следующем примере символ мнимой единицы введите с панели специальных символов

$(\mathbf{2}+i)(\mathbf{10}-i)$

$21+8i$

Для запоминания результатов вычислений удобно использовать переменные. При определении переменной (константы) используется знак равенства. Слева от знака равенства стоит имя переменной, а справа – ее значение или выражение для вычисления.

$$\boldsymbol{a}=\mathbf{5}^{\mathbf{3}}+\mathbf{4}\frac{\mathbf{12}+\mathbf{2}^{\mathbf{5}}}{\mathbf{2}^{\mathbf{4}}}$$

136

Такую константу можно использовать в различных выражениях:

$\boldsymbol{a}+\mathbf{7}$

143

$(\boldsymbol{a}-\mathbf{130})^{\mathbf{3}}$

216

Если длина формулы больше ширины окна документа, то система автоматически выполнит перенос части формулы на следующую строку. Никаких специальных знаков продолжения строки не используется.

Можно создавать свои функции. При определении функции необходимо использовать подчеркнутые символы после каждого имени аргумента в левой части и знак отложенного присваивания := (или обычного присваивания =) в середине. Справа от этого знака должно стоять выражение для вычисления значения функции.

$\boldsymbol{f}[\mathrm{x\_}]:=\boldsymbol{x}^{\mathbf{2}}$

Эту функцию можно вызывать с разными типами аргументов, например, с числовым аргументом

$\boldsymbol{f}[\mathbf{5}]$

25

Мы можем использовать ее с символьным выражением

$\boldsymbol{f}[\boldsymbol{c}+\boldsymbol{b}]$

$(\boldsymbol{b}+\boldsymbol{c})^{\mathbf{2}}$

Можно построить график этой функции

$\mathbf{Plot}[\boldsymbol{f}[\boldsymbol{x}],\{\boldsymbol{x},-\mathbf{2},\mathbf{2}\}]$

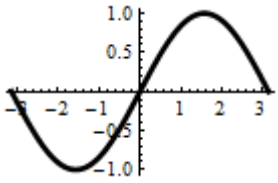
Можно взять производную этой функции

$\boldsymbol{f}'[\boldsymbol{u}]$

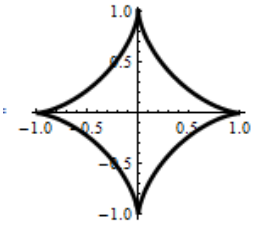
$2u$

Графические возможности пакета весьма обширны. Для первого знакомства мы приведем только несколько примеров. Наберите точно такие команды, и вы получите аналогичные рисунки

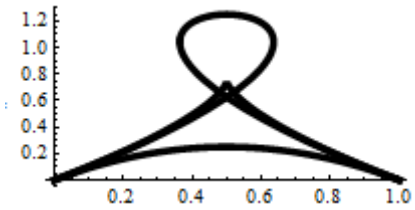
$\mathbf{Plot}[\mathbf{Sin}[\boldsymbol{x}],\{\boldsymbol{x},-\boldsymbol{\pi},\boldsymbol{\pi}\}]$



ParametricPlot[{**Sin**[t]^3, **Cos**[t]^3}, {t, **0**, **2Pi**}]



ParametricPlot[{
{t, $t - t^2$}, {$3t - 6t^2 + 4t^3$, $3t - 3t^2$},
{$5t - 12t^2 + 8t^3$, $5t - 5t^2$}}, {t, **0**, **1**},
AspectRatio → **0.5**, **PlotStyle** → {{**Black**, **Thickness**[**0.02**]}}]



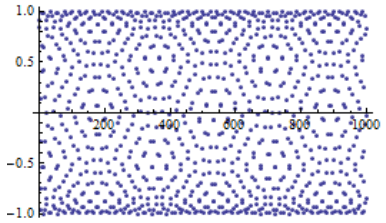
В последнем примере были построены три кривые, уравнения которых заданы в параметрическом виде (стрелочка набирается как знак минус и знак больше). В нашем тексте здесь и далее длинные команды занимают две или более строк. В *Mathematica* набирайте их одной строкой или выполняйте перенос на следующую строку нажатием клавиши Enter.

Для генерирования случайных чисел используется функция `Random[]`(по умолчанию в диапазоне от 0 до 1)

Random[ ]

Вот как можно построить график по некоторому набору точек

ListPlot[**Table**[**Sin**[i], {i, **1**, **1000**}]]

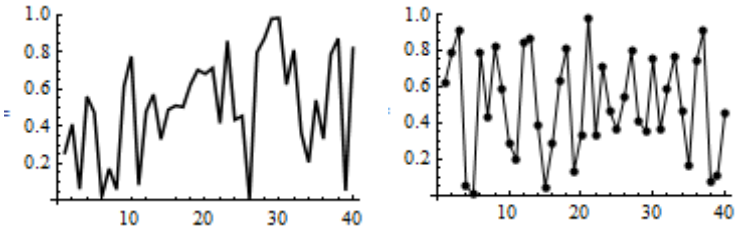


Если точки нужно соединять отрезками, то следует использовать опцию (необязательный аргумент) `Joined→True` (следующий рисунок слева).

ListPlot[**Table**[**Random**[], {**40**}], **Joined** → **True**,
PlotStyle → {{**Black**, **Thickness**[**0.01**]}}]

Здесь опция `Joined→True` повлияла на способ отображения графика – вместо точек была построена ломаная. Если вы хотите отобразить точки и ломаную, то следует добавить еще две опции Mesh → All и MeshStyle → PointSize[размер]. Например, следующий рисунок справа построен командой)

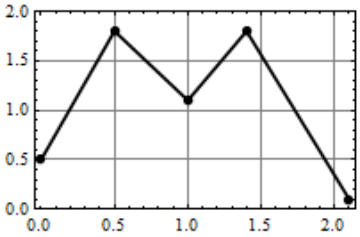
ListPlot[**Table**[**Random**[], {**40**}], **Joined** → **True**,
PlotStyle → {{**Black**, **Thickness**[**0.01**]}}, **Mesh** → **All**,
MeshStyle → **PointSize**[**0.03**]]



Заметим, что в Mathematica 5 опция, управляющая соединением точек, имела вид `PlotJoined→True`.

Вот еще один пример.

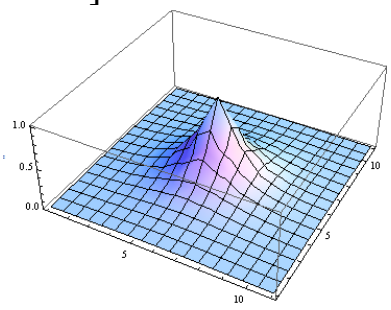
```
lst = {{0, 0.5}, {0.5, 1.8}, {1., 1.1}, {1.4, 1.8}, {2.1, 0.1}};
ListPlot[lst, Joined → True, Frame → True, PlotRange → {0., 2.},
    GridLines → Automatic, Mesh → All, MeshStyle → PointSize[0.03],
    PlotStyle → {{Black, Thickness[0.01]}}]
```



Здесь функции `ListPlot` в качестве аргумента передан список `lst`. Остальные аргументы (опции) не являются обязательными и передаются функции в виде: имя → значение. О них мы поговорим позже.

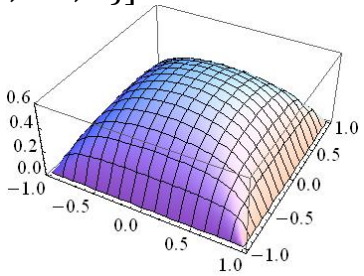
Вот пример табулирования точек поверхности

```
a1 = Table[1/(1 + x^2 + y^2), {x, -5, 5, 1}, {y, -5, 5, 1}];
ListPlot3D[a1, PlotRange → All]
```



Можно создавать функции двух переменных и строить поверхности по их уравнению

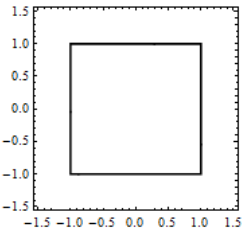
```
W[x_, y_] := 2 - x^2 - y^2 - Sqrt[(1 - x^2)^2 + (1 - y^2)^2]
Plot3D[W[x, y], {x, -1, 1}, {y, -1, 1}]
```



Здесь первым аргументом функции `Plot3D` является выражение, график которого следует построить, вторым и третьим – диапазоны изменения независимых переменных.

В следующем примере строится квадрат по его неявному уравнению W[x,y]=0, где выражение для W используется из предыдущего примера (в *Mathematica* это будет предыдущая секция документа). Обратите внимание на двойной знак равенства.

```
ContourPlot[W[x, y] == 0, {x, -1.5, 1.5}, {y, -1.5, 1.5}, PlotPoints → 100,
            ContourStyle → {{Black, Thickness[0.01]}}]
```



(В *Mathematica* 5 вместо функции `ContourPlot` использовалась функция `ImplicitPlot`, которая находится в пакете расширения `<<Graphics`ImplicitPlot`.` Также вместо опции `ContourStyle` использовалась опция `PlotStyle` с теми же значениями).

Многие функции системы *Mathematica* находятся в пакетах расширения – специальных программах, подключаемых во время работы. Чтобы выполнить такие функции надо загрузить соответствующий пакет командой << (подряд два знака меньше) с указанием имени пакета. Например, `<<ComputationalGeometry`.` Здесь используются обратная одинарная кавычка. Ее кнопка находится на клавиатуре под клавишей ESC. Для загрузки пакета можно также использовать функцию Needs. Например, Needs["ComputationalGeometry`"]. В этой записи используются двойные и обратная одинарная кавычки.

В следующем примере с помощью функции `PlanarGraphPlot[…]` строится триангуляция Делоне. Код функции находится в пакете расширения `ComputationalGeometry`` (обратная косая кавычка в конце имени обязательна). Координаты точек задаются в списке `data`.

```
<< ComputationalGeometry`
data = {{4.4, 14}, {6.7, 15.25}, {6.9, 12.8}, {2.1, 11.1}, {9.5, 14.9},
        {13.2, 11.9}, {10.3, 12.3}, {6.8, 9.5}, {3.3, 7.7}, {0.6, 5.1},
        {5.3, 2.4}, {8.45, 4.7}, {11.5, 9.6}, {13.8, 7.3}, {12.9, 3.1}, {11, 1.1}};
PlanarGraphPlot[data, LabelPoints → False]
```

Если надо использовать функции пакета расширения еще раз, то повторной загрузки пакета не требуется. Загруженные функции будут доступны в течении всего сеанса работы с текущим документом *Mathematica*.

Система символьных вычислений *Mathematica* как программа состоит из четырех основных частей – оболочки, ядра (*Kernel*), набора пакетов расширения и справочной системы. Оболочка – это интерфейс программы, которую мы видим на экране и которую используем для набора формул. Ядро – это динамически загружаемая библиотека функций, которою можно использовать даже из других программ, и функции которой выполняют все вычисления. Пакеты расширения содержат большую часть функциональных возможностей системы. Их может создавать любой пользователь.

Справочная система вызывается из меню Help – Documentation Center или клавишей F1. Если в документе выделить название какой-нибудь функции и нажать F1, то откроется окно со справкой по этой функции. Примеры можно выполнять в окне справки так же, как и в окне основного документа, нажатием комбинации клавиш Shift-Enter.

Работа с «Математикой» оформляется в виде отдельного файла, который имеет расширение имени *.nb. Файл разбит на ячейки, ячейки имеют стиль, который определяет, что можно делать с содержимым ячейки. Например, ячейки, содержащие вычисляемые выражения имеют стиль «Input», ячейки, содержащие результат, имеют стиль «Output». Бывают ячейки со стилем «Text» или с заголовочными стилями. Для выполнения вычислений в ячейке используется комбинация клавиш `Shift-Enter` или можно использовать клавишу `Enter` на цифровой области клавиатуры. После выполнения вычислений в секции после нее появляется «Панель предложения дальнейших действий», на которой мы можем выбрать что мы хотим сделать далее.

Двойной щелчок по графику и затем `Ctrl – d` откроет палитру рисования по графику.

Знакомство с другими возможностями системы вы найдете в следующих разделах данного пособия.

1. Основы алгебраических вычислений

1.1 Основные типы данных и переменные

В пакете `Mathematica` вы можете использовать любые знакомые вам из алгебры типы данных: целые числа, рациональные, иррациональные, комплексные. С основными из них вы познакомились в предыдущем разделе. Осталось показать, как работать с комплексными числами и переменными.

Для ввода комплексных чисел можно использовать стандартную нотацию, используя для мнимой единицы символ I (прописная буква I) или специальный символ $\boldsymbol{i}$, который можно ввести последовательной комбинацией клавиш Esc – **i** – **i** - Ecs (два раза символ i), а также с панели спецсимволов. Знаки операций используются те же, что и для вещественных чисел.

$$a = 2 + I; b = 3 - I; c = a * b$$
$$d = a/b$$
$7 + i$
$\frac{1}{2} + \frac{i}{2}$

Напомним, что точка с запятой в конце выражения отменяет вывод результатов расчета в окно документа. Поэтому значения переменных a и b при выводе не повторяются.

Функции `Re[z]` и `Im[z]` вычисляют вещественную и мнимую часть комплексного числа `z`.

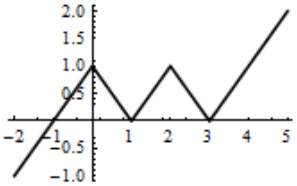
$$a = 2 + I; b = 3 - I; c = a * b$$
$$z = a + b * c - c/a$$
$21 - 2i$
$\mathbf{Re}[z]$
$\mathbf{Im}[z]$
21
-2

Функция `Abs[z]` и `Arg[z]` вычисляют модуль и аргумент комплексного числа `z`.

$\mathbf{Abs}[z]$
$\mathbf{Arg}[z]$
$\sqrt{445}$
$-\text{ArcTan}[\frac{2}{21}]$

Естественно, что функция `Abs[z]` используется и для вычисления модулей вещественных чисел.

$$\mathbf{Plot}[x - 1 - \mathbf{Abs}[x] + \mathbf{Abs}[x - 1] - \mathbf{Abs}[x - 2] + \mathbf{Abs}[x - 3], \{x, -2, 5\}]$$



Переменная может хранить любой тип данных – число, символ, список, матрицу, функцию, текст, графический фрагмент, звук и т.д. Для работы с ней можно использовать любые допустимые для такого типа данных функции. Например, любое имя, не имеющее никакого значения, интерпретируется системой как идентификатор (заменитель) числа и для работы с ним используются алгебраические функции. Любой набор имен можно перемежать знаками арифметических операций и выполнять над ним стандартные алгебраические преобразования.

Особое значение имеет операция присваивания. В системе имеется две операции: « = » – непосредственное присваивание и « := » – отложенное присваивание. В выражении `name=expr` происходит вычисление правой части выражения `expr` и сразу же результат присваивается переменной `name`. Вычисление правой части `expr` выражения отложенного присваивание

`name:=expr` выполняется в момент, когда потребовалось значение левой части, т.е. *name* потребовалось для вывода или вычисления. Для пояснения разницы рассмотрим следующую последовательность команд, содержащую непосредственное присваивание.

$\boldsymbol{a = 5; x = a^2; a = 6; x}$

25

Здесь мы присвоили значение 5 переменной *a*, затем переменной *x* присвоили значение a^2. В результате непосредственного присваивания произошло вычисление правой части выражения $x = a^2$ и левой части, т.е. переменной *x*, было присвоено значение 25. Последующее изменение значения переменной *a* не повлияло на значение *x*. Выполним туже последовательность команд, но для переменной *x* выполним отложенное присваивание (предварительно удалив прежнее значение *x* из рабочего пространства системы).

$\boldsymbol{x =.;\ a = 5;\ x := a^2; a = 6; x}$

36

Здесь присваивание $x = a^2$ выполнялось в момент вывода значения переменной *x*. Но в этот момент переменная *a* равнялась 6 и, поэтому, мы получили $x = 6^2$, т.е. 36.

Еще раз. Отложенное присваивание *x*:=f[*a*] означает, что значение f[*a*] не будет присвоено переменной *x* до тех пор, пока *x* не потребуется. Это означает, что в момент использования переменной *x* будет использовано последнее значение переменной *a*.

Кроме присваивания вам, наоборот, иногда потребуется удалять имена переменных. Для этого используется функция `Remove[x]`. Она удаляет из рабочего пространства системы символ *x* так, что его имя больше не распознается системой, в частности, не распознается тип переменной. Функция `Clear[x]` очищает значение и удаляет определение символа `x`. Удалить значение одной переменной можно командой `x=.` (`x` присвоить точку), а функциям `Remove` и `Clear` можно передавать имена нескольких переменных, перечисляемых через запятую.

Выполните следующие цепочки команд и проанализируйте результат.

$\boldsymbol{\mathbf{Clear}[f, x]; x = 3; f = x; x = 1; \{x, f\}}$

```
{1,3}
```

$\boldsymbol{\mathbf{Clear}[f, x]; f = x; x = 3; x = 1; \{x, f\}}$

```
{1,1}
```

$\boldsymbol{\mathbf{Clear}[f, x]; x = 3; f = x; x = 1; \mathbf{Clear}[x]; \{x, f\}}$

```
{x,3}
```

$\boldsymbol{\mathbf{Clear}[f, x]; f = x; x = 3; x = 1; \mathbf{Clear}[x]; \{x, f\}}$

```
{x,x}
```

$\boldsymbol{\mathbf{Clear}[f, x]; f = x; x = 3; x = 1; \mathbf{Remove}[x]; \{x, f\}}$

```
{Removed[x],Removed[x]}
```

Любой набор чисел или имен, заключенный в фигурные скобки, интерпретируется как список или даже вектор и над ним можно выполнять преобразования естественные для такого типа данных. В следующих разделах

пособия вы познакомитесь с этим и другими (нечисловыми) типами данных, и функциями для работы с ними.

1.2. Работа со списками, векторами и матрицами

В пакете *Mathematica* нет векторов и матриц, но есть списки, которые их заменяют. Любой набор элементов заключенный в фигурные скобки является списком. Список списков заменяет матрицу. Возможно и более глубокое вложение списков. Здесь мы рассматриваем основные операции, которые можно выполнять со списками.

Список можно складывать с числом. В этом случае число прибавляется к каждому элементу списка.

```
5+{10,20,30,40}
```

$\{15,25,35,45\}$

Список можно умножить на число.

```
5 {10,20,30,40}
```

$\{50,100,150,200\}$

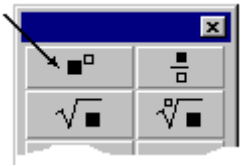
В этом случае каждый элемент списка умножается на соответствующее число.

Можно возвести в квадрат каждый элемент списка.

```
{10,20,30,40}^2
```

$\{100,400,900,1600\}$

Отметим, что возведение в степень можно выполнять с помощью операции ^ (шляпка) или с использованием специального трафарета панели специальных символов, который также можно ввести комбинацией клавиш Ctrl - ^.



Там, где может стоять число или переменная может стоять список.

```
Sin[{Pi/6, Pi/4, Pi/3, Pi/2}]
```

$$\left\{\frac{1}{2}, \frac{1}{\sqrt{2}}, \frac{\sqrt{3}}{2}, 1\right\}$$

Можно объединить вместе два списка.

```
Join[{10,20,30,40},{100,200,300,400}]
```

$\{10,20,30,40,100,200,300,400\}$

Мы можем получить элемент списка (двойные квадратные скобки обозначают извлечение элемента).

```
{10, 20, 30, 40}[[2]]
```

20

или

```
lst={10,20,30,40};
lst[[3]]
```

30

Можно выделить часть списка (промежуток)

$\boldsymbol{L = \{3, 5, 1, 2, -2, 0, 7, 8\}}$

$\boldsymbol{L[[3;;6]]}$

$\{1, 2, -2, 0\}$

Можно присваивать один список другому

$\boldsymbol{\{a, b, c\} = \{x^2, x\text{^}2 - 3x + 2, \frac{x+1}{x-1}\}}$

$\boldsymbol{a}$

$\boldsymbol{b}$

$\boldsymbol{c}$

x^2

$2 - 3x + x^2$

$$\frac{1 + x}{-1 + x}$$

При выполнении такого присваивания происходит вычисление выражений в правой части и затем присваивание результата элементам левого списка.

Можно сложить поэлементно два списка. В этом случае списки должны иметь одинаковое количество элементов.

{10,20,30,40}+{100,200,300,400}

```
{110,220,330,440}
```

Можно выполнить следующее сложение списков

{1,3,5}+{{2,1,3},{5,6,7},{3,2,3}}

```
{{3,2,4},{8,9,10},{8,7,8}}
```

Здесь каждый элемент первого списка складывается с соответствующим внутренним списком второго слагаемого.

Можно умножить поэлементно два списка. Это умножение не является скалярным произведением.

{10,20,30,40} {100,200,300,400}

```
{1000,4000,9000,16000}
```

Список может быть реорганизован:

Reverse[{10,20,30,40}] (обращение списка)

```
{40,30,20,10}
```

Sort[{4,2,3,1}] (сортировка списка по возрастанию)

```
{1,2,3,4}
```

Reverse[Sort[{4,2,3,1}]] (сортировка списка по убыванию)

```
{4,3,2,1}
```

Элементы списка не обязательно должны быть числами.

{x, x^2, x^3, x^4, Sin[x]}

Если элементами списка являются числа, то их можно представить графически.

$\boldsymbol{\mathbf{ListPlot}[\{4, 2, 3, 1, 5, 2, 5, 8, 4, 2, 4, 4, 2\}, \mathbf{PlotStyle} \rightarrow \mathbf{PointSize}[0.02]]}$

При построении графика этого списка абсциссами точек (узлов) являются числа {1,2,3,…} – номера элементов списка, а ординатами – значения элементов. Отметим, что точки, попавшие на оси координат незаметны из-за их размеров. Размер точек на графике управляется опцией PlotStyle → PointSize[0.02].

Можно организовать список списков. В общем случае можно представить список пар чисел:

$\{\{1,5\},\{4,2\},\{1,3\},\{3,3\},\{7,1\},\{5,5\},\{3,3\},\{2,5\}\}$

Такой список может функцией `ListPlot` будет интерпретироваться как список *x, y* координат точек.

ListPlot[%]

Напомним, что символ % представляет объект последнего вычисления системы, не предыдущий в документе, а последний вычисленный.

Существует много команд, связанных с обработкой списков. Вот некоторые из них: `First, Last, Rest, Part, Take, Drop, Append, Insert, Delete`.

$L = \{3,5,8,1,9,0,-2\}$

Part[L, 5] – выбирает 5-й элемент списка (=L[[5]]);

First[L] – выбирает первый элемент списка L;

Last[L] – выбирает последний элемент списка L;

Rest[L] – удаляет первый элемент списка L;

L[[{1, 3, 5}]] – возвращает список, составленный из элементов L с указанными номерами.

Смысл других функций понятен из их названия. Заметим, что эти функции не реорганизуют список L, а возвращают новый список. Например, функция `Rest[L]`, создает новый список, являющийся копией списка L без его первого элемента.

Получить краткую справку по любой функции можно, например, так:

```
?Delete
```

В результате в окне документа появится краткая информация об этой функции. Справку по функции можно открыть в отдельном окне справочной системы. Для этого надо внутри набранного в окне документа имени интересующей вас функции поместить курсор и нажать клавишу F1.

Списки можно создавать. Вот пример создания списка целых чисел.

```
Range[3,16,2]
{3,5,7,9,11,13,15}
```

Аргументами функции `Range` являются начальное значение списка, максимально допустимое значение и шаг.

Чтобы просуммировать элементы списка, можно использовать функцию `Sum`. Ее первым аргументом является формула вычисления общего члена, а вторым – список, указывающий имя изменяемой переменной и ее максимальное значение.

```
L:={1,5,3,4}
Sum[ L[[ i ]], {i,4}]
13
```

Вот еще примеры работы со списком. Введите в одной секции документа следующий код (без наших пояснений)

```
L1={1,2,4,3,8,6,7};
L2={{1,2,3},{4,5,6},{7,8,10}};
```

L1[[{1,3,5}]] - из списка `L1` выбрать 1-й, 3-й и 5-й элементы;

L2[[2,3]] - из L2 выбрать 2-й внутренний список и из него взять 3-й элемент;
Det[L2] - вычислить детерминант.
Вы увидите следующий результат:

```
{1,4,8}
 6
-3
```

Функция Length возвращает длину списка (число элементов верхнего уровня). Функция Dimensions возвращает список - количество элементов по каждому из индексов входного списка. Функция TensorRank возвращает количество индексов (размерность) списка.

Length[L1]
Dimensions[L1]
Dimensions[L2]
TensorRank[L2]

Поучаем следующий результат.

```
7
{7}
{3,3}
2
```

Имеются различные функции для автоматического создания списков. В следующем примере создается таблица квадратов целых чисел:

Table[n^2, {n, 1, 10}]

```
{1,4,9,16,25,36,49,64,81,100}
```

Первый аргумент - выражение для формирования элементов списка. Второй аргумент - «итерационная спецификация». Здесь она означает, что **n** изменяется от 1 до 10 с шагом 1. Мы можем изменить размер шага, добавив четвертый параметр в список:

Table[n^2, {n, 1, 10, ½}]

Двойной список, можно интерпретировать как прямоугольную матрицу и выполнять с ним естественные для таких объектов операции (транспонирование и т.д.).

e1=Table[{t, t^2,}, {t, 0, 5, 1}]

```
{{0,0}, {1,1}, {2,4}, {3,9}, {4,16}, {5,25}}
```

Transpose[e1]

```
{{0,1,2,3,4,5}, {0,1,4,9,16,25}}
```

Двойной список может быть реорганизован в одинарный.

Flatten[e1]

```
{0,0,1,1,2,4,3,9,4,16,5,25}
```

У функции Flatten есть вариант вызова Flatten$[L, n]$, где n представляет уровень реорганизации

$\boldsymbol{L = \{\{\{1,2,3\},\{4,5,6\},\{7,8,9\}\},\{\{1,1,0\},\{0,2,2\},\{0,0,0\}\}\}}$
$\mathbf{Flatten}[\boldsymbol{L}, \mathbf{0}]$
$\mathbf{Flatten}[\boldsymbol{L}, \mathbf{1}]$
$\mathbf{Flatten}[\boldsymbol{L}, \mathbf{2}]$
$\mathbf{Flatten}[\boldsymbol{L}]$

{{{1,2,3},{4,5,6},{7,8,9}},{{1,1,0},{0,2,2},{0,0,0}}}
{{1,2,3},{4,5,6},{7,8,9},{1,1,0},{0,2,2},{0,0,0}}
{1,2,3,4,5,6,7,8,9,1,1,0,0,2,2,0,0,0}
{1,2,3,4,5,6,7,8,9,1,1,0,0,2,2,0,0,0}
Как видим, вызов `Flatten` без второго аргумента эквивалентен вызову с максимальным значением уровня реорганизации.

Можно создавать список символьных выражений.

$$\mathbf{Table}[\mathbf{Expand}[(\mathbf{1}+\boldsymbol{x})^{\boldsymbol{n}}],\{\boldsymbol{n},\mathbf{1},\mathbf{3}\}]$$

$$\{1+x,1+2x+x^2,1+3x+3x^2+x^3\}$$

Мы можем создать список списков.

```
Table[{n, n^2, n^3, n^4}, {n, 1, 3}]
```

```
{{1,1,1,1},{2,4,8,16},{3,9,27,81}}
```

или

```
Table[n^a, {n, 1, 3}, {a, 1, 4}]
```

```
{{1,1,1,1},{2,4,8,16},{3,9,27,81}}
```

Первый аргумент - выражение, которое определяет каждый элемент списка. Второй аргумент - «медленный» итератор (изменяется при переходе от одного внутреннего списка к другому). Третий аргумент - «быстрый» итератор (изменяется при переходе от элемента к элементу внутри каждого внутреннего списка).

В пакете *Mathematica* имеется возможность префиксного (перед аргументом) и постфиксного (после аргумента) использования имен функций одной переменной. Если имя функции используется перед аргументом, то аргумент надо заключать в квадратные скобки, например, `Sin[Pi/4]`. Если имя функции стоит после аргумента, то аргумент отделяется от него двойной косой чертой, например, `Pi/4 // Sin`. Оба способа вернут одинаковый результат $\frac{1}{\sqrt{2}}$.

Функция `TableForm` показывает списки списков в двумерном виде. Ее удобно использовать в постфиксной нотации. Сравните

```
TableForm[{{a,b},{c,d}}]  или  {{a,b},{c,d}} // TableForm
```

```
a  b
c  d
```

Здесь запись `{{a,b},{c,d}} // TableForm` означает применение функции `TableForm` к списку `{{a,b},{c,d}}`.

Вот пример создания таблицы чисел 3×4.

$$\mathbf{TableForm}[\mathbf{Table}[\boldsymbol{n}^{\boldsymbol{a}},\{\boldsymbol{n},\mathbf{1},\mathbf{3}\},\{\boldsymbol{a},\mathbf{1},\mathbf{4}\}]]$$

```
1   1    1    1
2   4    8    16
3   9   27   81
```

Вектор в *Mathematica* - это список {x, y}. *Mathematica* не различает вектора-строки и вектора-столбцы. Список может быть и тем, и другим, в зависимости от контекста. Матрица в *Mathematica* - это список списков

$$\{\{\boldsymbol{a},\boldsymbol{b}\},\{\boldsymbol{c},\boldsymbol{d}\}\}$$

Чтобы увидеть более традиционную запись, можно добавить вызов функции MatrixForm.

MatrixForm[{{a,b},{c,d}}] или **{{a,b},{c,d}}// MatrixForm**

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

При вводе матриц добавление строк выполняется комбинацией клавиш Ctrl - Enter, а добавление столбцов Ctrl -,(запятая).

Функция Dot может выполнить матричное умножение списков (матриц), умножение матрицы на вектор или скалярное умножение векторов.

Dot[{{1,2},{3,4}},{{1,4},{2,5}}]

```
{{5,14},{11,32}}
```

Синонимом функции Dot является оператор '.' (точка). С его помощью мы можем определить произведение векторов и/или матриц.

{{a,b},{c,d}}.{x,y}

```
{ax+by,cx+dy}
```

или

{x,y}.{{a,b},{c,d}}

```
{ax+cy,bx+dy}
```

В первом случае {x,y} выступает, как вектор-столбец, во втором - как вектор-строка. В следующем выражении присутствуют оба типа.

$$\{x, y\}.\{x, y\}$$

$$x^2 + y^2$$

Можно использовать привычное обозначение векторов и матриц, но результат все равно будет списком списков. Если вы желаете его увидеть в виде вектора или матрицы используйте функцию MatrixForm.

$$\begin{pmatrix} x & y \end{pmatrix}.\begin{pmatrix} x \\ y \end{pmatrix}$$

$$\{\{x^2 + y^2\}\}$$

Для выполнения предыдущей команды наберите круглые скобки (обе, открывающую и закрывающую), внутри скобок наберите x, затем нажмите комбинацию клавиш Ctrl-запятая, затем введите y. Переместите курсов вправо за пределы скобок, введите точку, наберите снова открывающую и закрывающую скобки. Внутри этих скобок введите x, затем наберите комбинацию клавиш Ctrl-Enter, затем введите y. Аналогично выполняется набор следующей команды

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}.\begin{pmatrix} e \\ f \end{pmatrix}$$ **//MatrixForm**

$$\begin{pmatrix} a\,e + b\,f \\ c\,e + d\,f \end{pmatrix}$$

Рассмотрим некоторые встроенные функции для манипуляции векторами и матрицами. Все эти функции работают и с числовыми, и с символьными матрицами. Мы будем демонстрировать их на простых символьных матрицах, так что вы увидите результат.

Вычисление детерминанта

$\mathbf{Det}[\{\{\boldsymbol{a},\boldsymbol{b}\},\{\boldsymbol{c},\boldsymbol{d}\}\}]$

$-b\ c+a\ d$

Команда

$\mathbf{Det}[\mathbf{Table}[\boldsymbol{a}[\boldsymbol{i},\boldsymbol{j}],\{\boldsymbol{i},\mathbf{1},\mathbf{3}\},\{\boldsymbol{j},\mathbf{1},\mathbf{3}\}]]$

даст формулу вычисления детерминанта матрицы `3 x 3`

$$-a[1,3]a[2,2]a[3,1]+a[1,2]a[2,3]a[3,1]+a[1,3]a[2,1]a[3,2]-a[1,1]a[2,3]a[3,2]-a[1,2]a[2,1]a[3,3]+a[1,1]a[2,2]a[3,3]$$

Можно вычислить след матрицы

$\boldsymbol{M}=\mathbf{Table}[\boldsymbol{a}[\boldsymbol{i},\boldsymbol{j}],\{\boldsymbol{i},\mathbf{1},\mathbf{3}\},\{\boldsymbol{j},\mathbf{1},\mathbf{3}\}];$

$\boldsymbol{M}\mathbf{//MatrixForm}$

$\mathbf{Tr}[\boldsymbol{M}]$

$$\begin{pmatrix} a[1,1] & a[1,2] & a[1,3] \\ a[2,1] & a[2,2] & a[2,3] \\ a[3,1] & a[3,2] & a[3,3] \end{pmatrix}$$

a[1,1]+a[2,2]+a[3,3]

Матрицу можно транспонировать

$\mathbf{MatrixForm}[\mathbf{Transpose}[\{\{\boldsymbol{a},\boldsymbol{b}\},\{\boldsymbol{c},\boldsymbol{d}\}\}]]$

$$\begin{pmatrix} a & c \\ b & d \end{pmatrix}$$

Можно найти обратную матрицу (если она существует).

$\boldsymbol{A}=\mathbf{Inverse}\begin{bmatrix}\mathbf{1} & \mathbf{2}\\ \mathbf{3} & \mathbf{4}\end{bmatrix}$

$$\{\{-2,1\},\{\frac{3}{2},-\frac{1}{2}\}\}$$

$\mathbf{MatrixForm}[\boldsymbol{A}]$

$$\begin{pmatrix} -2 & 1 \\ \frac{3}{2} & -\frac{1}{2} \end{pmatrix}$$

Аналогично

$\mathbf{Inverse}\left[\begin{pmatrix}\boldsymbol{a} & \boldsymbol{b}\\ \boldsymbol{c} & \boldsymbol{d}\end{pmatrix}\right]\mathbf{//MatrixForm}$

$$\begin{pmatrix} \frac{d}{-bc+ad} & -\frac{b}{-bc+ad} \\ -\frac{c}{-bc+ad} & \frac{a}{-bc+ad} \end{pmatrix}$$

Имеются функции для автоматического создания матриц некоторых типов. На самом деле такие функции создают список списков, который мы можем интерпретировать как матрицу. Функция `IdentityMatrix` создает единичную матрицу указанного размера. Функция `DiagonalMatrix` создает диагональную матрицу. Функция `Array[f,{n1,n2}]` генерирует $n_1 \times n_2$ массив – двойной список с элементами `f[i1,i2]`. Здесь имя `f` интерпретируется как имя функции, например, `Sin`. Вот несколько примеров.

MatrixForm[**IdentityMatrix**[**3**]]

$$\begin{pmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{pmatrix}$$

MatrixForm[**DiagonalMatrix**[{$\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$}]]

$$\begin{pmatrix}a & 0 & 0\\ 0 & b & 0\\ 0 & 0 & c\end{pmatrix}$$

MatrixForm[**Array**[$\boldsymbol{a}$, {**3**, **3**}]]

$$\begin{pmatrix}a[1,1] & a[1,2] & a[1,3]\\ a[2,1] & a[2,2] & a[2,3]\\ a[3,1] & a[3,2] & a[3,3]\end{pmatrix}$$

или, например,

$\boldsymbol{f}$[**x_**, **y_**]: = $\boldsymbol{x}\ \boldsymbol{y}$

Array[$\boldsymbol{f}$, {**3**, **3**}]//**MatrixForm**

$$\begin{pmatrix}1 & 2 & 3\\ 2 & 4 & 6\\ 3 & 6 & 9\end{pmatrix}$$

Здесь мы создали функцию двух переменных *f(x,y)* и с ее помощью генерировали матрицу размера 3x3.

Функции `Eigenvalues` и `Eigenvectors` вычисляют собственные числа и собственные вектора матрицы.

Eigenvalues[$\begin{pmatrix}\mathbf{1} & \mathbf{4}\\ \mathbf{2} & \mathbf{3}\end{pmatrix}$]

{-1,5}

Eigenvectors[$\begin{pmatrix}\mathbf{1} & \mathbf{4}\\ \mathbf{2} & \mathbf{3}\end{pmatrix}$]

{{-2,1},{1,1}}

Напомним, что скалярное произведение векторов вычисляется с использованием операции «точка».

{1,2}.{3,4}

11

Векторное произведение векторов вычисляется функцией `Cross`.

Cross[{1,2,3},{2,4,5}]

```
{-2,1,0}
```

Cross[{1,2,3,2},{2,4,5,1},{1,1,1,1}]

```
{-3,5,-3,1}
```

Векторное произведение `n` векторов в `n+1` мерном пространстве даст вектор ортогональный ко всем сомножителям. Ортогональность можно проверить с помощью скалярного произведения результирующего вектора на каждый вектор сомножитель.

При решении систем линейных уравнений полезной является функция приведения матрицы к диагональному виду (если это возможно). Функция `RowReduce[Matr]` выполняет приведение матрицы к ступенчатой форме методом Гаусса. При этом невырожденная матрица будет приводиться к единичной.

$$\mathbf{RowReduce}\left[\begin{pmatrix}\mathbf{1} & \mathbf{2} & \mathbf{3}\\ \mathbf{4} & \mathbf{5} & \mathbf{6}\\ \mathbf{7} & \mathbf{8} & \mathbf{8}\end{pmatrix}\right]\mathbf{//MatrixForm}$$

$$\begin{pmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{pmatrix}$$

$$\mathbf{RowReduce}\left[\begin{pmatrix}\mathbf{1} & \mathbf{2} & \mathbf{3}\\ \mathbf{4} & \mathbf{5} & \mathbf{6}\end{pmatrix}\right]\mathbf{//MatrixForm}$$

$$\begin{pmatrix}1 & 0 & -1\\ 0 & 1 & 2\end{pmatrix}$$

$$\mathbf{RowReduce}\left[\begin{pmatrix}\mathbf{1} & \mathbf{2} & \mathbf{3}\\ \mathbf{4} & \mathbf{5} & \mathbf{6}\\ \mathbf{7} & \mathbf{8} & \mathbf{9}\end{pmatrix}\right]\mathbf{//MatrixForm}$$

$$\begin{pmatrix}1 & 0 & -1\\ 0 & 1 & 2\\ 0 & 0 & 0\end{pmatrix}$$

В последнем примере у ненулевой вырожденной матрицы последняя строка приводится к нулевой, а последний столбец будет ненулевым (если он не был нулевым с самого начала).

1.3 Алгебраические преобразования

Свое название «Системы аналитических вычислений» получили в связи их умением выполнять различные преобразования алгебраических выражений. Для того, чтобы выполнить какое-либо преобразование выражения, его (выражение) надо передать аргументом соответствующей функции. В этом параграфе мы рассмотрим примеры таких функций.

Функция `Factor` раскладывает алгебраическое выражение на множители.

$$\mathbf{Factor}[\boldsymbol{a^2 + 2ab + b^2}]$$

$$(a+b)^2$$

или

$$\mathbf{Factor}[\mathbf{6 + 11}\boldsymbol{x} \mathbf{- 3}\boldsymbol{x^2} \mathbf{- 2}\boldsymbol{x^3}]$$

$$-(-2+x)(3+x)(1+2x)$$

Можно ли x^4+4 разложить на множители без комплексных чисел. Давайте посмотрим, что скажет Mathematica:

$$\mathbf{Factor}[\boldsymbol{x^4} \mathbf{+ 4}]$$

$$(2-2x+x^2)(2+2x+x^2)$$

Если мы хотим разрешить комплексные числа при разложении на множители, то мы можем использовать следующий вариант:

$$\mathbf{Factor}[\boldsymbol{x^2} \mathbf{+ 9, GaussianIntegers} \rightarrow \mathbf{True}]$$

$$(-3i+x)(3i+x)$$

`GaussianIntegers->True` - это так называемая опция (необязательный аргумент функции): «Я хочу разрешить комплексные числа с целыми коэффициентами в результате».

По умолчанию *Mathematica* раскладывает на множители с целыми коэффициентами, поэтому следующий пример не раскладывается.

$$\mathbf{Factor}[x^2 - 3]$$

Mathematica не раскладывает на множители «в радикалах». Разложение на множители «в радикалах» - это не разложение в обычном смысле слова; это больше похоже на отыскание корней уравнений, которое делается командой `Solve` (см. следующий параграф):

$$\mathbf{Solve}[x^2 - 3 == 0, x]$$

$$\{\{x \to -\sqrt{3}\}, \{x \to \sqrt{3}\}\}$$

Однако, функция `Factor` имеет опцию `Extension`, использование которой изменяет поведение алгоритма разложения на множители.

$$\mathbf{Factor}[1 + x^4]$$

$$1 + x^4$$

$$\mathbf{Factor}[1 + x^4, \mathbf{Extension} \to \sqrt{2}]$$

$$-(-1 + \sqrt{2}x - x^2)(1 + \sqrt{2}x + x^2)$$

$$\mathbf{Factor}[1 + x^4, \mathbf{Extension} -> \{\sqrt{2}, I\}]$$

$$\frac{1}{4}(\sqrt{2} - (1 + i)x)(\sqrt{2} - (1 - i)x)(\sqrt{2} + (1 - i)x)(\sqrt{2} + (1 + i)x)$$

$$\mathbf{Factor}[2 + 2\sqrt{2}x + x^2]$$

$$2 + 2\sqrt{2}x + x^2$$

$$\mathbf{Factor}[2 + 2\sqrt{2}x + x^2, \mathbf{Extension} -> Automatic]$$

$$(\sqrt{2} + x)^2$$

Использование опции `GaussianIntegers→True` эквивалентно использованию опции `Extension→I`.

$$\mathbf{Factor}[1 + x^2, \mathbf{GaussianIntegers} -> True]$$

$$(-i + x)(i + x)$$

$$\mathbf{Factor}[1 + x^2, \mathbf{Extension} -> I]$$

$$(-i + x)(i + x)$$

Есть еще одна полезная опция `Trig→True`, которая позволяет разложить в произведение (если это возможно) тригонометрические выражения.

$$\mathbf{Factor}[\mathbf{Sin}[2x] + \mathbf{Sin}[2y], \mathbf{Trig} \to \mathbf{True}]$$

$$2\mathrm{Cos}[x - y]\mathrm{Sin}[x + y]$$

Если вы раскладываете на множители полиномы, включающие дроби, то *Mathematica* приводит все к общему знаменателю

$$\mathbf{Factor}\left[x^2 - \frac{4}{9}\right]$$

$$\frac{1}{9}(-2 + 3x)(2 + 3x)$$

Mathematica может раскладывать на множители выражения, содержащие функции вместо простых переменных:

$$\mathbf{Factor}[f[x]^2 - g[x]^2]$$

$$(f[x] - g[x])(f[x] + g[x])$$

Функция Expand обратная к функции Factor. Она умеет раскрывать скобки в алгебраическом выражении.

$$\mathbf{Expand}[(\boldsymbol{a}+\boldsymbol{b})^{\mathbf{5}}]$$

$$a^5+5a^4b+10a^3b^2+10a^2b^3+5ab^4+b^5$$

В формате Expand[expr, patt] функция оставляет не разложенной любую часть выражения expr, которая не содержит выражения patt.

$$\mathbf{Expand}[(\boldsymbol{a}+\boldsymbol{b})^{\mathbf{2}}(\mathbf{1}+\boldsymbol{x})^{\mathbf{2}}]$$

$$a^2+2ab+b^2+2a^2x+4abx+2b^2x+a^2x^2+2abx^2+b^2x^2$$

$$\mathbf{Expand}[(\boldsymbol{a}+\boldsymbol{b})^{\mathbf{2}}(\mathbf{1}+\boldsymbol{x})^{\mathbf{2}},\boldsymbol{x}]$$

$$(a+b)^2+2(a+b)^2x+(a+b)^2x^2$$

В следующем примере раскрываются скобки только у выражения вида 1+x

$$\mathbf{Expand}[(\mathbf{1}+\boldsymbol{x})^{\mathbf{3}}+(\mathbf{2}+\boldsymbol{x})^{\mathbf{4}}+(\mathbf{1}+\boldsymbol{y})^{\mathbf{2}},\mathbf{1}+\boldsymbol{x}]$$

$$1+3x+3x^2+x^3+(2+x)^4+(1+y)^2$$

В следующем примере остаются нетронутыми элементы выражения, которые не соответствуют шаблону x[_]

$$\mathbf{Expand}[(\boldsymbol{a}+\boldsymbol{b})(\boldsymbol{x}[\mathbf{1}]+\boldsymbol{x}[\mathbf{2}])^{\mathbf{2}},\boldsymbol{x}[\_]]$$

$$(a+b)x[1]^2+2(a+b)x[1]x[2]+(a+b)x[2]^2$$

Опция Trig→True подсказывает, что нужно выполнять тригонометрические разложения

$$\mathbf{Expand}[\mathbf{Sin}[\boldsymbol{x}+\boldsymbol{y}],\mathbf{Trig}\rightarrow\mathbf{True}]$$

$$\mathrm{Cos}[y]\mathrm{Sin}[x]+\mathrm{Cos}[x]\mathrm{Sin}[y]$$

Без этой опции разложение не выполняется

$$\mathbf{Expand}[\mathbf{Sin}[\boldsymbol{x}+\boldsymbol{y}]]$$

$$\mathrm{Sin}[x+y]$$

Функция Together выполняет приведение к общему знаменателю:

$$\mathbf{Together}\left[\frac{\boldsymbol{a}}{\boldsymbol{b}}+\frac{\boldsymbol{c}}{\boldsymbol{d}}\right]$$

$$\frac{b\,c+a\,d}{b\,d}$$

Наоборот, функция Apart выполняет разложение дробей:

$$\mathbf{Apart}\left[\frac{\mathbf{b\,c+a\,d}}{\mathbf{b\,d}}\right]$$

$$\frac{a}{b}+\frac{c}{d}$$

В выражении собирание коэффициентов при одинаковых степенях переменной, определяемой вторым аргументом, можно выполнить с помощью функции Collect:

$$\mathbf{Collect}[\boldsymbol{a}\,\boldsymbol{x}+\boldsymbol{b}\,\boldsymbol{x}+\boldsymbol{c}\,\boldsymbol{y}+\boldsymbol{d}\,\boldsymbol{y},\boldsymbol{x}]$$

$$(a+b)x+cy+dy$$

и

$$\mathbf{Collect}[\boldsymbol{a}\,\boldsymbol{x}+\boldsymbol{b}\,\boldsymbol{x}+\boldsymbol{c}\,\boldsymbol{y}+\boldsymbol{d}\,\boldsymbol{y},\{\boldsymbol{x},\boldsymbol{y}\}]$$

$$(a+b)x+(c+d)y$$

Функции TrigExpand и TrigReduce выполняют действия, аналогичные Expand и Collect применительно к тригонометрическим выражениям.

TrigExpand[**Sin**[3x]]

$3\text{Cos}[x]^2\text{Sin}[x] - \text{Sin}[x]^3$

TrigReduce[**Sin**[x]**Cos**[2y]]

$\frac{1}{2}(\text{Sin}[x-2y] + \text{Sin}[x+2y])$

Функция `TrigToExp` преобразует тригонометрические и гиперболические функции в экспоненты

TrigToExp[**Sin**[2x]]

$\frac{1}{2}i e^{-2ix} - \frac{1}{2}i e^{2ix}$

TrigToExp[**ArcTanh**[x]]

$-\frac{1}{2}\text{Log}[1-x] + \frac{1}{2}\text{Log}[1+x]$

Функция `ExpToTrig` выполняет действия, обратные `TrigToExp`.

ExpToTrig[**Exp**[$I x$]]

$\text{Cos}[x] + i\text{Sin}[x]$

ExpToTrig[**Sqrt**[I]]

$\frac{1+i}{\sqrt{2}}$

Упрощение алгебраических выражений выполняется функцией `Simplify`.

Simplify $\left[\frac{x^2+2xy+y^2}{x+y}\right]$

x+y

`Simplify[expr]` и `FullSimplify[expr]` – приводят выражение `expr` к наиболее простому виду. `Simplify` пробует найти простейшую форму выражения. Часто бывает не совсем ясно, какая из форм будет проще, и разные люди могут иметь различные мнения о простоте. *Mathematica* считает простейшим выражение с наименьшим количеством элементов.

Simplify[**Sin**[x]^2 + **Cos**[x]^2]

1

Однако часто она не может ничего сделать с выражением и поэтому вам надо вызывать самому функции преобразования (`Factor`, `Expand` и др.), чтобы выполнить упрощение.

Функция `FullSimplify` умеет выполнять упрощения со многими специальными функциями.

Для выполнения упрощений с учетом дополнительных условий используется второй аргумент этих функций

Simplify[**Sqrt**[x^2], **Element**[x, **Reals**]]

Abs[x]

То же можно выполнить командой

Simplify[**Sqrt**[x^2], $x \in$ **Reals**]

Здесь символ ∈ представляет инфиксную (между аргументами) запись функции двух переменных `Element[x,Reals]`.

Вот пример вызова функции `Simplify` без каких – либо предположений

Simplify[**Sqrt**[x**^2**]]

$\sqrt{x^2}$

Тогда

Simplify[**Sqrt**[x**^2**], $x > 0$]

x

или

Simplify[**Sqrt**[x**^2**], $x \in$ **Reals**]

Abs[x]

Аналогично

Simplify[**Sqrt**[x**^2**y**^2**], {$x \in$ **Reals**, $y < 0$}]

$-y$Abs[x]

Второй аргумент можно записывать, используя опцию `Assumptions`.

Simplify[**Sqrt**[x**^2**]/x + **Sqrt**[y**^2**]/y, **Assumptions** → {$x > 0, y > 0$}]

2

1.4. Решение алгебраических уравнений

Большинство алгебраических преобразований мы можем выполнить самостоятельно. Однако решение уравнений является боле сложной задачей. И в этом вопросе помощь *Mathematica* может быть значительной. Здесь мы рассматриваем функции пакета, с помощью которых можно решать алгебраические уравнения. В системе реализованы все известные методы «точного» определения корней уравнений. Но даже в тех случаях, когда соответствующих формул нет, мы можем найти приближенное значение корня (корней).

Для символьного решения уравнения используется функция `Solve`

Solve[$x^2 - 2x ==$ **0**, x]

{{$x \to 0$}, {$x \to 2$}}

Первый аргумент - это уравнение, которое должно быть решено. Второй аргумент - переменная, относительно которой мы решаем уравнение. Результат возвращается в форме списка правил подстановки («замены») с использованием операции ‘→’ стрелка (минус и знак больше). В данном случае показано, что решение *x* равно 0 или 2. Вот пример решения кубического уравнения

$e = x^3 - 2x + 1 ==$ **0**

Solve[e, x]

$$\left\{\{x \to 1\}, \left\{x \to \frac{1}{2}\left(-1 - \sqrt{5}\right)\right\}, \left\{x \to \frac{1}{2}\left(-1 + \sqrt{5}\right)\right\}\right\}$$

Помните, что имена неизвестных, фигурирующих в уравнениях, не должны содержать значений. Если, например, переменной `x`, используемой в примере, ранее было присвоено значение, то его (значение) надо «забыть». Попробуйте, например, выполнить команды

$\boldsymbol{x = 5}$;

$\mathbf{Solve}[\boldsymbol{x^2 - 3x + 2 == 0}, \boldsymbol{x}]$

```
Solve::ivar: _5_ is not a valid variable>>
```

Solve[False, 5]

Вы получаете сообщение об ошибке и результат Solve[False,5]. Что произошло? Перед решением алгебраического уравнения вы присвоили переменной *x* значение 5. Система подставила в выражение $x^2 - 3x + 2$ вместо *x* значение 5, получила значение 12, которое не совпадает с 0. В результате, вместо ожидаемого уравнения $x^2 - 3x + 2 == 0$ первый аргумент функции `Solve` равен False.

Чтобы такого не происходило, искомое имя (в данном случае *x*) не должно содержать значения. Для этого перед вызовом функции, решающей уравнение или систему уравнений, следует вызвать функцию `Remove[x]` или `Clear[x]`. Функциям `Remove` и `Clear` можно передавать имена нескольких переменных, перечисляемых через запятую.

Замечание по «очистке» имен переменных касается всех примеров этого параграфа. Если перед кодом примера вы будете добавлять команду `Remove[имя1, имя2, ...]` , где `имя1, имя2,` это имена переменных, используемых в коде, то в большинстве случаев этого достаточно для корректного выполнения текущего примера.

Иное по сравнению с `Solve` представление решения дает функция `Roots`.

$\mathbf{rts} = \mathbf{Roots}[\boldsymbol{e}, \boldsymbol{x}]$

$x == \frac{1}{2}\left(-1 - \sqrt{5}\right) \;||\; x == \frac{1}{2}\left(-1 + \sqrt{5}\right) \;||\; x == 1$

Второе представление можно преобразовать к первому с помощью функции `ToRules`

$\mathbf{ToRules}[\mathbf{rts}]$

$\text{Sequence}[\{x \to \frac{1}{2}(-1 - \sqrt{5})\}, \{x \to \frac{1}{2}(-1 + \sqrt{5})\}, \{x \to 1\}]$

Если уравнения содержат другие переменные, то они трактуются как константы

$\mathbf{Solve}[\boldsymbol{x^2 - 5xy + 4y^2 == 0}, \boldsymbol{x}]$

$\{\{x \to y\}, \{x \to 4y\}\}$

Вы можете решать системы уравнений, содержащие более чем одну неизвестную, используя для этого список уравнений и список неизвестных

$\mathbf{Solve}[\{\boldsymbol{2x + 3y == 13, 3x - 2y == 0}\}, \{\boldsymbol{x, y}\}]$

$\{\{x \to 2, y \to 3\}\}$

Первый аргумент - список решаемых уравнений. Второй аргумент - список переменных, относительно которых решаются уравнения.

Приведенная выше система имеет одно решение (одна пара значений *x* и *y*). Если существует более одного решения, то будет получен список списков значений

$\mathbf{Solve}[\{\boldsymbol{x^2 + y^2 == 16, x^2 - 4 == y}\}, \{\boldsymbol{x, y}\}]$

$\{\{x \to 0, y \to -4\}, \{x \to -\sqrt{7}, y \to 3\}, \{x \to \sqrt{7}, y \to 3\}\}$

Первый элемент результата $\{x \to 0, y \to -4\}$ - это первое решение и т.д. Добавляя команду `//TableForm` к команде `Solve`, мы делаем вывод более читабельным

$$\mathbf{Solve}[\{x^2 + y^2 == 16, x^2 - 4 == y\}, \{x, y\}]\mathbf{//TableForm}$$

$x \to 0$ $\quad y \to -4$

$x \to -\sqrt{7}$ $\quad y \to 3$

$x \to \sqrt{7}$ $\quad y \to 3$

Каждая строка представляет одно решение. Обратите внимание на кратный корень.

Если решения не существует, то возвращается пустой список:

$$\mathbf{Solve}[\{x^2 - 4 == 0, x^2 - 3 == 0\}, x]$$

{ }

Решение уравнений функцией Solve возвращается в форме списка правил подстановки. Поясним эти правила на примере.

$$x^2/.x \to 5$$

25

Здесь сказано, что надо заменить *x* на 5 в выражении x^2. Оператор /. (слеш и точка) читается «заменить» и → (стрелка) читается как «на». Все выражение следует читать «в x^2 заменить *x* на 5».

Оператор /. (слеш и точка) является постфиксной формой функции `ReplaceAll`

$$\mathbf{ReplaceAll}[x^2, x{-}{>}\, 5]$$

Можно использовать список правил замены более чем одной переменной за один раз

$$x^2 + y^2/.\{x \to 5, y \to 10\}$$

125

Когда вы используете список списков правил замены, вы получаете список результатов

$$x^2/.\{\{x \to 5\}, \{x \to 6\}\}$$

```
{25, 36}
```

или

$$x^2 + y^2/.\{\{x \to 5, y \to 10\}, \{x \to 6, y \to 10\}\}$$

```
{125, 136}
```

Вернемся к первому примеру решения уравнения

$$s = \mathbf{Solve}[x^2 - 2x == 0, x]$$

$\{\{x \to 0\}, \{x \to 2\}\}$

Мы получили решение в виде списка списков правил подстановки, и можем использовать его для замены значений в любых выражениях. Например, мы можем проверить, что ответ правильный, подставляя решение в исходное уравнение

$$x^2 - 2x/.s$$

{0, 0}

Список из двух нулей означает, что оба решения дают нулевые значения, когда мы подставляем их в исходное выражение.

$$x^2 - 3x /. s[[2]]$$

-2

т.к. `s[[2]]={x->2}`, и

$$x /. s[[2]]$$

2

Можно прямо использовать команду `Solve` для замены в выражении

$$x^2 + 1 /. \text{Solve}[x^2 - 2x == 0, x]$$

```
{1,5}
```

или

$$x /. \text{Solve}[x^3 + ax^2 + ax + 1 == 0, x][[2]]$$

$$\frac{1}{2}(1 - a - \sqrt{-3 - 2a + a^2})$$

Запись `Solve[x^3+a x^2 + a x+1==0,x][[2]]` означает выбор второго элемента списка правил подстановки, т.е. второго корня уравнения.

Для получения списка значений решений удобно использовать следующую форму

$$x /. \text{Solve}[x^2 - 4x + 3 == 0, x]$$

```
{1,3}
```

Некоторые уравнения (например, полиномы 5 степени и выше) не могут быть решены в явном виде

$$s = \text{Solve}[x^6 + x^5 + x^2 + 1 == 0, x]$$

$$\{\{x \to \text{Root}[1 + \#1^2 + \#1^5 + \#1^6\&,1]\},$$
$$\{x \to \text{Root}[1 + \#1^2 + \#1^5 + \#1^6\&,2]\},$$
$$\{x \to \text{Root}[1 + \#1^2 + \#1^5 + \#1^6\&,3]\},$$
$$\{x \to \text{Root}[1 + \#1^2 + \#1^5 + \#1^6\&,4]\},$$
$$\{x \to \text{Root}[1 + \#1^2 + \#1^5 + \#1^6\&,5]\},$$
$$\{x \to \text{Root}[1 + \#1^2 + \#1^5 + \#1^6\&,6]\}\}$$

Здесь функция Root[f,k] представляет «точное значение» k – го корня полиномиального уравнения f[x]==0. Под «точным значением» понимается величина, которая может быть вычислена с любой степенью точности. Например

$$N[s[[1]]]$$

$$\{x \to -1.1540824 - 0.613722i\}$$

или

$$N[s[[1]], 20]$$

$$\{x \to -1.1540824762585313148 - 0.61372296835498784283i\}$$

Фактически, мы сразу могли написать

$$s = N[\text{Solve}[x^6 + x^5 + x^2 + 1 == 0, x]]$$

и получить *приближенные* значения всех шести корней. В то же время, функция $\text{Root}[1 + \#1^2 + \#1^5 + \#1^6\&, \text{k}]$ представляет «точное» значение.

Применение функции `N[Solve[…]]` эквивалентно вызову функции `NSolve[…]` с теми же аргументами. Функция `NSolve` находит приближенно все корни алгебраического (полиномиального) уравнения.

NSolve[**1** + **2***x* + **3***x*^**2** + **4***x*^**3** == **0**, *x*]

{{*x* → −0.60583}, {*x* → −0.07208 − 0.6383*i*}, {*x* → −0.07208 + 0.6383*i*}}

Функция `NRoots` находит приближенные решения полиномиальных уравнений, но ответ представляет по – другому. Например, решение предыдущей задачи она представляет в виде уравнений

NRoots[**1** + **2***x* + **3***x*^**2** + **4***x*^**3** == **0**, *x*]

x == −0.60583 || *x* == −0.0721 − 0.6383 *i* || *x* == −0.0721 + 0.6383 *i*

При желании преобразовать последнее решение в список правил подстановки можно выполнить команду

{**ToRules**[%]}

{{*x* → −0.60583}, {*x* → −0.07208 − 0.6383*i*}, {*x* → −0.07208 + 0.6383*i*}}

Если уравнение содержит параметр, то его корень будет зависеть от этого параметра. Тогда для построения функции (значение корня в зависимости от параметра) можно поступить так

s = **First**[**Solve**[***x*^2** − **2***x* − ***a*^4** == **0**, *x*]]

z[**t_**] = (***x*** /. ***s***) /. ***a*** → ***t***

{***z***[***x***], ***z***′[***x***], ***z***[**3**], ***z***′[**3**]}

$$\{1-\sqrt{1+x^4},-\frac{2x^3}{\sqrt{1+x^4}},1-\sqrt{82},-27\sqrt{\frac{2}{41}}\}$$

Построенная в этом примере функция *z*[*t*] является «настоящей» функцией – мы вычислили ее значение в точке и взяли производную. В этом примере задание функции с использованием отложенного присваивания в виде

z[**a_**] := ***x*** /. ***s***

некорректно, т.к. уже команда

z[x]

дает неверный ответ. Но сработает непосредственное присваивание

Remove[*x*, *s*, *z*, *a*, *t*]

s = **First**[**Solve**[***x*^2** − **2***x* − ***a*^4** == **0**, *x*]]

z[**a_**] = ***x*** /. ***s***

{***z***[***x***], ***z***′[***x***], ***z***[**3**], ***z***′[**3**]}

Если нужно использовать отложенное присваивание, то можно использовать функцию `Evaluate`

z[**a_**] := **Evaluate**[***x*** /. ***s***]

Как мы видели, функция `Solve` позволяет находить все корни уравнения или системы уравнений (если это возможно).

Для решения линейных систем уравнений можно использовать функцию `LinearSolve`. Функция `LinearSolve[M,b]` находит вектор *x*, который удовлетворяет матричному уравнению `M·x=b`, где `M` – матрица коэффициентов, `b` – вектор свободных членов.

$\mathbf{LinearSolve}\begin{bmatrix}\mathbf{1} & \mathbf{2} & \mathbf{2}\\ \mathbf{3} & \mathbf{4}, & \mathbf{3}\end{bmatrix}$

$\{\{-1\},\{\frac{3}{2}\}\}$

Если решение не единственно, то возвращается только одно решение.

$\boldsymbol{s} = \mathbf{LinearSolve}[\boldsymbol{m} = \begin{pmatrix}\mathbf{1} & \mathbf{2} & \mathbf{3}\\ \mathbf{2} & \mathbf{3} & \mathbf{4}\end{pmatrix}, \boldsymbol{v} = \{-\mathbf{1}, \mathbf{3}\}]$

```
{9,-5,0}
```

Заметим, что присваивания $m = \begin{pmatrix}1 & 2 & 3\\ 2 & 3 & 4\end{pmatrix}$ и $v = \{-1,3\}$, использованные в предыдущем примере, являются операциями, возвращающими результат. Результатом операций присваивания являются выражения стоящие в правой части, т.е. матрица $\begin{pmatrix}1 & 2 & 3\\ 2 & 3 & 4\end{pmatrix}$ и список $\{-1,3\}$. Т.о. за одно обращение к функции `LinearSolve` мы выполнили присвоение переменных m и v , а также корректную передачу параметров.

В этом примере мы должны были бы получить бесконечное множество решений. Для получения общего решения в подобных случаях надо использовать функцию `Solve`.

$\mathbf{Solve}[\boldsymbol{m}.\{\boldsymbol{a},\boldsymbol{b},\boldsymbol{c}\} == \boldsymbol{v}, \{\boldsymbol{a},\boldsymbol{b},\boldsymbol{c}\}]$

```
Solve::svars: Equations may not give solutions for all
"solve" variables.
```

$\{\{b \to 13 - 2a, c \to -9 + a\}\}$

При этом мы получили предупреждающее сообщение о том, что функция `Solve` не уверена, что смогла определить все решения системы. Обратите также внимание на способ записи системы уравнений в этом примере. Она имеет вид

$\boldsymbol{m}.\{\boldsymbol{a},\boldsymbol{b},\boldsymbol{c}\} == \boldsymbol{v}$

$\{a + 2b + 3c, 2a + 3b + 4c\} == \{-1,3\}$

Если решение системы не существует, то выводится соответствующее сообщение, а также повторяется ввод.

$\mathbf{LinearSolve}[\begin{pmatrix}\mathbf{1} & \mathbf{2}\\ \mathbf{3} & \mathbf{4}\\ \mathbf{5} & \mathbf{6}\end{pmatrix}, \{-\mathbf{1}, \mathbf{2}, -\mathbf{3}\}]$

```
LinearSolve::nosol: Linear equation encountered that has
no solution.
```

$\mathrm{LinearSolve}[\{\{1,2\},\{3,4\},\{5,6\}\},\{-1,2,-3\}]$

При решении системы с использованием функции `LinearSolve[M,b]` аргумент b может быть вектором или матрицей. Если он матрица, то определяются решения, соответствующие каждому столбцу матрицы b.

$\boldsymbol{s} = \mathbf{LinearSolve}[\boldsymbol{m} = \begin{pmatrix}\mathbf{1} & \mathbf{2} & \mathbf{3}\\ \mathbf{2} & \mathbf{3} & \mathbf{4}\\ \mathbf{3} & \mathbf{1} & \mathbf{3}\end{pmatrix}, \boldsymbol{v} = \begin{pmatrix}\mathbf{7} & \mathbf{1}\\ \mathbf{12} & \mathbf{2}\\ \mathbf{7} & \mathbf{3}\end{pmatrix}]$

```
{{2,1},{4,0},{-1,0}}
```

Здесь первое решение дается тройкой чисел `{2,4,-1}`, а второе – тройкой `{1,0,0}`.

Далеко не всегда функции *Mathematica* могут найти решение. Иногда им надо помочь, выполнив предварительно некоторые преобразования уравнения или системы уравнений.

Функция `Reduce[eqns,vars]` упрощает уравнения, заданные списком `eqns`, и пытается решить их относительно переменных, заданных списком `vars`. Уравнения, возвращаемые функцией `Reduce`, эквивалентны первоначальным и содержат все возможные решения. Результирующие уравнения объединяются с использованием операций логического ‘И’ (&&) или логического ‘ИЛИ’ (||).

$\mathbf{Reduce}[\{\boldsymbol{x}+\boldsymbol{y}+\boldsymbol{z}==\mathbf{1},\boldsymbol{x}-\boldsymbol{y}-\boldsymbol{z}==\mathbf{0}\},\{\boldsymbol{x}\}]$

$y==\frac{1}{2}-z\ \&\&\ x==\frac{1}{2}$

$\mathbf{Reduce}[\{\boldsymbol{x}+\boldsymbol{y}+\boldsymbol{z}==\mathbf{1},\boldsymbol{x}-\boldsymbol{y}-\boldsymbol{z}==\mathbf{0}\},\{\boldsymbol{x},\boldsymbol{z}\}]$

$x==\frac{1}{2}\ \&\&\ z==\frac{1}{2}-y$

$\mathbf{Reduce}[\{\boldsymbol{a}\boldsymbol{x}+\boldsymbol{y}+\boldsymbol{z}==\mathbf{1},\boldsymbol{x}-\boldsymbol{y}-\boldsymbol{z}==\mathbf{0}\},\{\boldsymbol{x},\boldsymbol{y}\}]$

$1+a\neq 0\ \&\&\ x==\frac{1}{1+a}\ \&\&\ y==x-z$

В формате вызова `Reduce[eqns,vars,elims]` функция упрощает уравнения и пытается исключить неизвестные `elims`.

$\mathbf{Reduce}[\{\boldsymbol{a}\,\boldsymbol{x}+\boldsymbol{y}+\boldsymbol{z}==\mathbf{1},\boldsymbol{x}-\boldsymbol{y}-\boldsymbol{z}==\mathbf{0}\},\{\boldsymbol{y}\},\{\boldsymbol{a}\}]$

$y==x-z\ \&\&\ x\neq 0$

(решаем относительно *y* и исключаем *a*)

Функция `Eliminate[eqns,vars]` упрощает систему уравнений путем исключения неизвестных `vars`. Она работает хорошо с линейными и полиномиальными уравнениями.

$\mathbf{Eliminate}[\{\boldsymbol{x}^{\mathbf{2}}==\mathbf{2}+\mathbf{2}\boldsymbol{x}+\boldsymbol{y},\boldsymbol{x}+\boldsymbol{y}==\boldsymbol{z}\},\boldsymbol{y}]$

$z==-2-x+x^2$

Здесь переменная *y* исключена.

$\mathbf{Eliminate}[\{\boldsymbol{a}\boldsymbol{x}^{\mathbf{2}}+\boldsymbol{b}\boldsymbol{x}\boldsymbol{y}+\boldsymbol{c}\boldsymbol{y}^{\mathbf{2}}==\mathbf{0},\boldsymbol{x}+\boldsymbol{y}==\mathbf{0}\},\boldsymbol{x}]$

$cy^2==(-a+b)y^2$

Здесь исключена переменная *x*. В результате получено уравнение относительно неизвестной y.

Функция `SolveAlways[eqns, vars]` возвращает список значений параметров, подстановка которых превращает уравнение в тождество относительно переменных `vars`.

$\mathbf{SolveAlways}[\boldsymbol{a}\,\boldsymbol{x}+\boldsymbol{b}==\mathbf{0},\boldsymbol{x}]$

$\{\{a\to 0,b\to 0\}\}$

$\mathbf{SolveAlways}[\boldsymbol{a}\boldsymbol{x}^{\mathbf{3}}+\boldsymbol{x}(\boldsymbol{x}-\boldsymbol{b})^{\mathbf{2}}+\boldsymbol{c}\boldsymbol{x}==\mathbf{0},\boldsymbol{x}]$

$\{\{a\to -1,c\to 0,b\to 0\}\}$

$\mathbf{SolveAlways}[\boldsymbol{a}\boldsymbol{x}^{\mathbf{3}}+(\boldsymbol{x}-\boldsymbol{b})^{\mathbf{2}}+\boldsymbol{c}==\mathbf{0},\boldsymbol{x}]$

{ }

Используя эту функцию, можно вывести теорему Виета для полиномов 2 – й и 3 – й степени.

$\mathbf{SolveAlways}[x^2 + ax + b == (x - \mathrm{x1})(x - \mathrm{x2}), x]$

$\{\{a \to -\mathrm{x1} - \mathrm{x2}, b \to \mathrm{x1x2}\}\}$

$\mathbf{SolveAlways}[x^3 + bx^2 + cx + d == (x - \mathrm{x1})(x - \mathrm{x2})(x - \mathrm{x3}), x]$

$\{\{b \to -\mathrm{x1} - \mathrm{x2} - \mathrm{x3}, c \to \mathrm{x1x2} + \mathrm{x1x3} + \mathrm{x2x3}, d \to -\mathrm{x1x2x3}\}\}$

Функция `FindRoot[eqn, {x,x0}]` пытается найти численное решение уравнения `eqn` методом Ньютона, используя начальное приближение `x=x0`.

$\mathbf{FindRoot}[x^2 == \mathbf{Exp}[x + 2], \{x, -1\}]$

$\{x \to -1.37015\}$

$\mathbf{FindRoot}[x^5 + 3e^{-x^4} == 0, \{x, -1\}, \mathbf{WorkingPrecision} \to 40]$

$\{x \to -1.010909816596691797197753835847306190493\}$

Помните, что определяется один ближайший к начальному приближению корень.

$\mathbf{FindRoot}[\mathbf{Cos}[x]^4 + 2\mathbf{Cos}[x]^2 == 0, \{x, 0.1\}]$

$\{x \to 4.71239\}$

Выбирая другое начальное приближение, мы получаем другой корень

$\mathbf{FindRoot}[\mathbf{Cos}[x]^4 + 2\mathbf{Cos}[x]^2 == 0, \{x, -1\}]$

$\{x \to -1.5708\}$

Вот что происходит, если функция FindRoot не может найти решение.

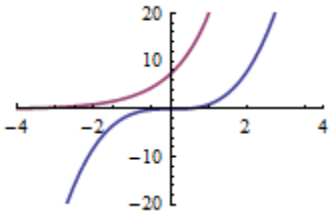
$\mathbf{FindRoot}[x^3 == \mathbf{Exp}[x + 2], \{x, -1\}]$

```
FindRoot::lstol: The line search decreased the step size
to within tolerance specified by AccuracyGoal and
PrecisionGoal but was unable to find ...
```

$\{x \to -0.967488\}$

Если построить график функций, стоящих в левой и правой части уравнения, то станет понятно, что решения не существует.

$\mathbf{Plot}[\{x^3, \mathbf{Exp}[x + 2]\}, \{x, -4, 4\}, \mathbf{PlotStyle} \to \mathbf{Thickness}[0.01],$
$\mathbf{PlotRange} \to \{\{-4, 4\}, \{-20, 20\}\}]$



Если использовать комплексное число в качестве начального приближения, то функция `FindRoot` вернет комплексный корень (ближайший к начальному приближению).

$\mathbf{FindRoot}[x^3 == \mathbf{Exp}[x + 2], \{x, -1 + i\}]$

$\{x \to -1.02483 + 0.930313\, i\}$

Используя функцию `FindRoot` можно находить приближенные решения систем уравнений.

$\mathbf{FindRoot}[\{x^2 + y^2 + x + 2y - 1 == 0, x^2\mathbf{Exp}[x] - y == 0\}, \{x, 0.1\}, \{y, 0.5\}]$

```
{x→0.381748,y→0.213474}
```

Имеется несколько необязательных аргументов (опций) у функции `FindRoot`. Опция `AccuracyGoal->m` определяет точность (количество цифр m)

результата. Опция `WorkingPrecision->n` определяет точность (количество цифр n) внутренних вычислений.

$$\mathbf{FindRoot}[x^3 == \mathbf{Exp}[x+2]-5, \{x,-1\}, \mathbf{AccuracyGoal} \to 24, \mathbf{WorkingPrecision} \to 44]$$

$$\{x \to -1.49532716056162153436189242485345399849506 06\}$$

Если окажется, что шагов итерации недостаточно для получения требуемой точности, то их количество можно увеличить с помощью опции `MaxIterations->p`.

$$\mathbf{FindRoot}[x^4 == 0, \{x, 0.023\}, \mathbf{AccuracyGoal} \to 24, \mathbf{WorkingPrecision} \to 34]$$

После выполнения приведенного кода вы получите сообщение, что процесс определения корня остановился после 15 итераций и требуемая точность не достигнута. Тогда увеличьте число итераций следующим образом:

$$\mathbf{FindRoot}[x^4 == 0, \{x, 0.023\}, \mathbf{AccuracyGoal} \to 24, \mathbf{WorkingPrecision} \to 34, \mathbf{MaxIterations} \to 200]$$

$$\{x \to 2.357817533643951878386887880936 35 \times 10^{-24}\}$$

С другими, реже используемыми функциями решения алгебраических уравнений, вы сможете познакомиться по справочной системе.

1.5. Функции и выражения в системе Mathematica.

В системе имеется огромное количество встроенных функций, однако имеется возможность создавать свои функции. С несколькими простыми способами мы уже познакомились. Простейший способ состоит в выполнении команды

$y[x_] := $ *выражение*

или

$y[x\_] =$ *выражение* (без двоеточия)

Здесь *y* – имя функции (такое как Sin для функции Sin[x]). Выражение в правой части представляет любой набор операций и вызовов функций, использующих переменную *x* (без символа подчеркивания). Символ подчеркивания в левой части говорит о том, что переменная *x* в правой части выражения является локальной переменной. Знак **:=** представляет отложенное присваивание, которое выполняется в тот момент, когда происходит конкретное вычисление значения функции. Если используется знак = , то выполняется создание функции немедленно. Вы должны понимать разницу между термином «функция» и «выражение». Например, Sin – это функция, а Sin[x] – выражение. Аналогично выше мы создали функцию y, а y[x] – это выражение.

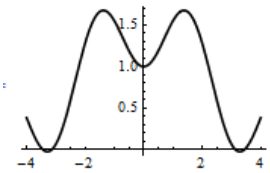
После создания функцию можно использовать. Например, можно вычислять значения функций в точке или строить график.

$$y[x\_] = \mathbf{Sin}[x]^2 + \frac{\mathbf{Sin}[x]}{x};$$

$$\{y[0.0001], N[y[1]], N[y[\frac{\pi}{2}]]\}$$

$$\{1., 1.54954, 1.63662\}$$

$\mathbf{Plot}[\boldsymbol{y}[\boldsymbol{x}], \{\boldsymbol{x}, -\mathbf{4}, \mathbf{4}\}]$



Обратите внимание, что при построении графика используется выражение y[x]. Такую функцию можно дифференцировать (на самом деле мы дифференцируем выражение y[x], получаемое из функции).

$\boldsymbol{D}[\boldsymbol{y}[\boldsymbol{x}], \boldsymbol{x}]$

$\frac{\mathrm{Cos}[x]}{x} - \frac{\mathrm{Sin}[x]}{x^2} + 2\mathrm{Cos}[x]\mathrm{Sin}[x]$

При создании функции можно использовать другие функции

$\mathbf{ex}[\mathbf{x\_}] := \mathbf{Expand}[(\mathbf{1} + \boldsymbol{x})^{\mathbf{2}}]$

$\mathbf{ex}[\boldsymbol{x}]$

$1 + 2x + x^2$

Операция «;» – точка с запятой объединяет несколько выражений в одно так, что результатом является результат последнего выражения в цепочке. Тогда функцию можно создавать так

$\boldsymbol{y}[\mathbf{x\_}] = (\boldsymbol{b} = \boldsymbol{x} + \boldsymbol{x}^{\mathbf{2}}; \boldsymbol{c} = \boldsymbol{x} + \boldsymbol{b}; \boldsymbol{c} + \boldsymbol{b});$

$\boldsymbol{y}[\boldsymbol{x}]$

$3x + 2x^2$

О любой функции, встроенной или созданной пользователем, системе можно задать вопрос. Для этого надо набрать имя функции, перед которым поставить вопросительный знак.

? **Sin**

Sin[z] gives the sine of z.>>

? **ex**

Global`ex

$\mathrm{ex}[\mathrm{x\_}] := \mathrm{Expand}[(1 + x)^2]$

Более подробную информацию о функции можно получить, если поставить два вопросительных знака перед именем функции.

? ? **Sin**

Sin[z] gives the sine of z.>>

$\mathrm{Attributes}[\mathrm{Sin}] = \{\mathrm{Listable}, \mathrm{NumericFunction}, \mathrm{Protected}\}$

? ? $\boldsymbol{y}$

Global`y

$y[\mathrm{x\_}] = 3x + 2x^2$

Общий способ создания функции пользователя состоит в использовании функции `Function`.

```
funcname = Function[x, body]
```

Эта форма предназначена для определения простой функции с одним формальным параметром (переменной) *x*. Здесь `funcname` имя функции, такое как Sin, Log и т.д., *body* – выражение, определяющее правило вычисления. Например

$\boldsymbol{z = \mathbf{Function}[x, x^2]};$

$\boldsymbol{z[2]}$

4

Здесь z – имя функции, первое вхождение x в правой части – имя формального параметра, затем следует выражение, описывающее правило вычисления значения функции. Выражение может состоять из нескольких выражений, разделенных точкой с запятой. Результат вычисления последнего выражения в такой последовательности будет результатом вычисления функции.

$\boldsymbol{\mathbf{f1} = \mathbf{Function}[x, g = x^2; g * \mathbf{Sin}[x]]};$

$\boldsymbol{\mathbf{Plot}[\mathbf{f1}[x], \{x, 0, 4\}]}$

С функцией, созданной таким образом, можно работать как с любой другой. Например, такую функцию можно дифференцировать

$\boldsymbol{D[\mathbf{f1}[x], x]}$

$x^2\mathrm{Cos}[x] + 2x\mathrm{Sin}[x]$

При создании можно не определять имя функции.

$\boldsymbol{\mathbf{Function}[x, x^2]};$

Тогда можно использовать знак % вместо имени функции (% представляет результат выполнения последней операции в системе).

$\%[\boldsymbol{n}]$

n^2

Функцию можно определить без указания имен ее аргументов. Тогда для обращения к ним следует использовать значки #1, #2 и т.д.

`funcname=Function[body]`

где `funcname` имя функции, *body* – выражение, определяющее правило вычисления. Например,

$\boldsymbol{\mathbf{f2} = \mathbf{Function}[\#1^2]}$

$\#1^2\&$

Здесь #1 заменяет имя первого (и единственного) аргумента. Тогда в выражениях можно использовать `f2[x]` или вместо *x* подставлять конкретные значения `f2[2]`.

Синонимом основного определения функции является ее постфиксное определение в виде

$\boldsymbol{\mathbf{funcname} = (\#1^2)\&}$

$\#1^2\&$

Формальные параметры при определении функции обозначаются # (или #1, #2 и т.д.). Само имя функции обозначается #0, ## используется для обозначения всего списка аргументов, ##n обозначает списка аргументов, начиная с n-того. Значок **&** используется вместо ключевого слова `Function` после указания тела функции. Таким образом, последнее определение эквивалентно постфиксному использованию `Function`.

$$\mathbf{funcname} = \overbrace{\left(\#1^2\right)}^{\&}\text{//Function}$$

Следующие примеры иллюстрируют применение описанных правил и обозначений.

$\mathbf{Function}[x, x^2][5]$ вернет 25.

$(\# + 1)\&[x]$ вернет 1+x.

$F[\#\#, \#\#2]\&[x, y, z]$ даст $F[x, y, z, y, z]$

Вероятно, последняя команда вернет другой результат, поскольку в текущей сессии работы системы *Mathematica* мы переменным y и z присваивали некоторые значения. Чтобы система забыла определение переменных можно, например, выполнить команду

$\mathbf{Remove}[x, y, z, f, F]$

где в список «забываемых» переменных мы на всякий случай включили и другие имена. После этого повторите команду $F[\#\#, \#\#2]\&[x, y, z]$.

Для определения функции с несколькими аргументами можно использовать форму

`Function[{x1, x2,…},body]`

с использованием списка формальных параметров `{x1,x2,…}`. Можно использовать и простую форму, например

$z[x\_, y\_] = x^2 + y^2;$

$D[z[x, y], x]$

$2x$

$\mathbf{Plot3D}[z[x, y], \{x, -1, 1\}, \{y, -1, 1\}]$

$z[2, 3]$

13

Есть еще один способ создания функции с использованием символа отображения (стрелки)

$z = x \to x^3$

$\mathrm{Function}[x, x^3]$

Здесь стрелка особая! Чтобы правильно ее набрать надо последовательно нажать комбинацию клавиш $Esc - fn - Esc$.

Можно создавать рекуррентные функции. Например, функция факториала может быть создана следующим образом

$f = \mathbf{If}[\#1 == 1,\ 1, \#1 * \#0[\#1 - 1]\ ]\&;$

$f[5]$

120

Здесь в определении функции #1 заменяет имя первого (и единственного) аргумента функции, а #0 – используется вместо имени функции `f`. Таким образом, запись $\#0[\#1 - 1]$ означает вызов той же функции `f`, но с аргументом на единицу меньшим.

В *Mathematica* есть несколько способов применить функцию к выражению. Это обычные квадратные скобки f[x], префикс f@x и постфикс x // f. Например,

$\{\mathbf{Sin}\left[\frac{\pi}{6}\right],\ \mathbf{Sin@}\frac{\pi}{4},\ \frac{\pi}{3}\ //\mathbf{Sin}\}$

$\left\{\frac{1}{2}, \frac{1}{\sqrt{2}}, \frac{\sqrt{3}}{2}\right\}$

Префиксное и постфиксное использование имени функции допустимо для функций одной переменной. Для функции двух переменных возможно инфиксное (между аргументами) использование имени функции. Например

$\boldsymbol{f}[\mathbf{x\_}, \mathbf{y\_}] = \boldsymbol{x} + \boldsymbol{y}^{\mathbf{2}}$

$2 \sim f \sim 3$

11

или

$1 \sim f \sim 2 \sim f \sim 3$

14

Для некоторых функций двух переменных имеются специальные обозначения при использовании в инфиксной форме. Например, в уравнениях используется знак ==, являющийся инфиксной формой функции Equal. Знак → является инфиксной формой функции Rule.

Вам часто придется обращаться к элементам выражений, которые создает система, и потребуется выделять части выражений. К элементам выражений мы можем обращаться как к элементам списков по индексу. Например

$\boldsymbol{z} = \boldsymbol{f}[\boldsymbol{g}[\boldsymbol{a}], \boldsymbol{h}[\boldsymbol{b}]]$

$\{\boldsymbol{z}[[\mathbf{0}]], \boldsymbol{z}[[\mathbf{1}]], \boldsymbol{z}[[\mathbf{2}]], \boldsymbol{z}[[\mathbf{2}, \mathbf{0}]], \boldsymbol{z}[[\mathbf{2}, \mathbf{1}]]\}$

$\{f, g[a], h[b], h, b\}$

Здесь для выделения элементов выражения `z` мы используем индексацию. Так `z[[0]]=f` – имя внешней функции; `z[[1]]=g[a]` – первый аргумент функции, `z[[2]]=h[b]` – второй аргумент; `z[[2,0]]=h` – имя функции, стоящей во втором аргументе; `z[[2,1]]=b` – аргумент второй функции.

Например, создадим функцию

$\boldsymbol{f} = \mathbf{Function}[\{\boldsymbol{x}\}, \boldsymbol{t} = \boldsymbol{x}^{\mathbf{3}}; \boldsymbol{t} + \mathbf{1}];$

Тогда

$\{\boldsymbol{f}[[\mathbf{0}]], \boldsymbol{f}[[\mathbf{1}]], \boldsymbol{f}[[\mathbf{2}]]\}$

$\{\text{Function}, \{x\}, 1 + x^3\}$

Таким образом, обращаясь к имени f , используя индексацию, мы выделяем элементы в правой части выражения. Вот еще один пример.

$\boldsymbol{s} = \mathbf{Factor}[\boldsymbol{x}^{\mathbf{4}} - \mathbf{3}^{\mathbf{4}}]$

$(-3 + x)(3 + x)(9 + x^2)$

$\{\boldsymbol{s}[[\mathbf{1}]], \boldsymbol{s}[[\mathbf{2}]], \boldsymbol{s}[[\mathbf{3}]]\}$

$\{-3 + x, 3 + x, 9 + x^2\}$

или

$\boldsymbol{p} = \mathbf{Factor}[\boldsymbol{x}^{\mathbf{5}} - \mathbf{2}^{\mathbf{5}}]$

$(-2 + x)(16 + 8x + 4x^2 + 2x^3 + x^4)$

$\mathbf{Last}[\boldsymbol{p}]$

$16 + 8x + 4x^2 + 2x^3 + x^4$

Создание функций является ответственным моментом и скорее относится к разделу программирования. Здесь мы дали только введение в способы создания и использования функций.

2. Графические возможности системы.

В системе *Mathematica* реализовано огромное количество графических функций. В данном разделе представлен обзор основных. Предполагается, что читатель выполняет приведенные команды в системе *Mathematica*, поэтому иногда результаты их выполнения (графики) не приводятся.

2.1 Двумерные графики.

2.1.1 Обыкновенные графики функций.

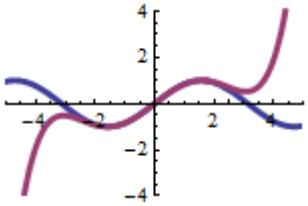
График функции, заданной явным уравнением `y=f(x)`, строится с помощью функции `Plot`.

$$\mathbf{Plot}[x^2, \{x, -1, 1\}]$$

Первый аргумент – выражение `f(x)` (правая часть уравнения), график которого надо построить. Второй аргумент определяет интервал изменения переменной *x*. *Mathematica* автоматически выбирает масштаб по вертикальной оси и выбирает достаточное количество точек, чтобы кривая была гладкой. Все эти параметры можно изменять с помощью опций. Если вы рисуете график функции с большими значениями, такой как тригонометрическая функция `Tan[x]`, *Mathematica* пробует подобрать разумные вертикальные пределы.

Можно нарисовать два и более графика в одних и тех же осях, используя список функций, как первый аргумент команды `Plot`

$$\mathbf{Plot}[\{\mathbf{Sin}[x], x - x^3/6 + x^5/120\}, \{x, -5, 5\}]$$



Можно изменить цвет и/или штриховой узор кривой, используя опцию `PlotStyle`. Следующая кривая имеет правильный штриховой узор.

$$\mathbf{Plot}[\mathbf{Sin}[x], \{x, -7, 7\}, \mathbf{PlotStyle} \to \mathbf{Dashing}[\{0.05\}]]$$

Штрихи и пробелы имеют длину 0.05 ширины графика. Если у вас есть более чем одна кривая на одном графике, вы может различить их по цвету и/или штриховому узору. В следующем примере первый элемент списка `PlotStyle` применяется к первому графику, второй – ко второму. Функция `Hue[...]` определяет цвет; ее аргумент должен изменяться в диапазоне от 0 до 1.

$$\mathbf{Plot}[\{\mathbf{Sin}[x], x - x^3/6 + x^5/120\}, \{x, -7, 7\}, \mathbf{PlotStyle} \to \{\mathbf{Hue}[2/3], \mathbf{Hue}[1/3]\}]$$

Можно пометить график, используя опцию `PlotLabel`

$$\mathbf{Plot}[\mathbf{Sin}[x], \{x, 0, 2\mathbf{Pi}\}, \mathbf{PlotLabel} \to \text{"The Sine Function"}]$$

Любой текст в Mathematica должен быть заключен в двойные кавычки.

Можно пометить оси графика, используя опцию `AxesLabel`

$$\mathbf{Plot}[\mathbf{Sin}[x], \{x, 0, 2\mathbf{Pi}\}, \mathbf{AxesLabel} \to \{\text{"x"}, \text{"Sin[x]"}\}]$$

Можно поместить весь график в рамку

$$\mathbf{Plot}[\mathbf{Sin}[x], \{x, 0, 2\mathbf{Pi}\}, \mathbf{Frame} \to \mathbf{True}]$$

Можно нарисовать координатную сетку

Plot[Sin[x], {x, 0, 2Pi}, GridLines → Automatic, Frame → True]

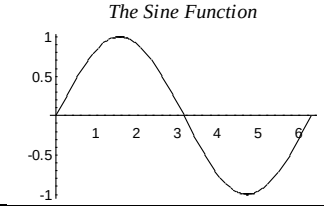
Можно изменить шрифты, используемые для текстовых элементов графиков.

В *Mathematica* 5 и более ранних версиях для этого использовалась опция `DefaultFont`. Шрифты задаются, как список: первый элемент – имя шрифта, второй – размер шрифта.

Plot[Sin[x], {x, 0, 2Pi}, DefaultFont → {"Helvetica − Oblique", 18}]

Для изменения шрифта индивидуального текстового элемента (метки, надписи), используют функцию `FontForm`. Она имеет два аргумента: первый – текст выражения, второй – спецификация шрифта.

Plot[Sin[x], {x, 0, 2Pi}, PlotLabel → FontForm["The Sine Function",
{"Palatino − Italic", 18}], DefaultFont → {"Helvetica", 14}]



В *Mathematica* 6 использовались опции `TextStyle` и `StyleForm`. В *Mathematica* 9 для изменения начертания текстовых элементов используются опция `BaseStyle` и функция `Style`.

Plot[Sin[x], {x, 0, 2Pi},
PlotLabel → Style["The Sine Function", {"Palatino − Italic", 18}],
BaseStyle → {"Helvetica", 14}] (* см. след. рисунок слева *)

Для изменения шрифта индивидуального текстового элемента (метки, надписи), используется функция `Style`, а для задания базового стиля других текстовых элементов используется опция `BaseStyle`.

Plot[Sin[x]^2, {x, 0, 2π}, PlotLabel → "Квадрат синуса",
BaseStyle → {FontWeight → "Bold", FontSize−> 18,
FontFamily → "Arial", FontSlant → "Italic"}](* след. рисунок в центре *)

Опция `LabelStyle` используется для изменения внешнего вида меток.

Plot[Sin[x], {x, 0, 10}, LabelStyle → {Blue, 16}] (* пред. рис. справа *)

Тот же график строится следующей командой

Plot[Sin[x], {x, 0, 10}, LabelStyle → Directive[Blue, 16]]

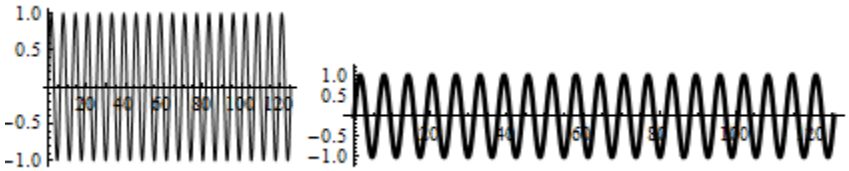
Любой текст, выводимый в пределах графической области, можно включить в тело функции `Style` первым аргументом и последующими аргументами изменить его внешний вид. Следующие несколько примеров демонстрируют возможности функции `Style` (часто функцию `Style` со всеми ее аргументами включают в тело функции `Text`).

```
Style["Hello Kharkov", Blue, Italic, 24]
```
Hello Kharkov
```
Table[Style["AaBbCc", Large, p], {p, {Bold, Italic, Underlined, Purple}}]
```
{**AaBbCc**, *AaBbCc*, AaBbCc, AaBbCc}
```
Style["Mathematica", "Subtitle"]
```
Mathematica
```
Table[Style[Sqrt[Abs[x]], Background → c],
      {c, {LightYellow, LightBlue, LightRed, LightGreen}}]
```
{√Abs[x], √Abs[x], √Abs[x], √Abs[x]}
```
Style[1/Sqrt[2], FontColor → Red, Background → LightGray]
```
$\frac{1}{\sqrt{2}}$
```
Table[Style["AaBbCcDd", FontFamily → p], {p, {"Courier", "Times", "Helvetica"}}]
```
{AaBbCcDd, AaBbCcDd, AaBbCcDd}
```
Table[Text[Style["Kharkov", FontSize → p]], {p, {12, 24, 36}}]
```
{Kharkov, Kharkov, Kharkov}
```
Table[Style[Sqrt[2], Magnification → m], {m, {1/2, 1, 2}}]
```
{√2, √2, √2}

Имеется и другие опции функции `Style`, с которыми вы можете познакомиться по справочной системе.

Mathematica обычно строит графики с отношением длина/ширина $\frac{1}{2}(1+\sqrt{5})$, что приблизительно равно `1.61803` (известно, как Золотое Сечение). Изменить экранное соотношение высоты и ширины графической области можно с помощью опции `AspectRatio`. Например, следующий график слева выглядит густовато

```
Plot[Sin[x], {x, 0, 40Pi}]
```



Это будет выглядеть лучше, если график немного растянуть (т.е. уменьшить отношение высота/ширина)

```
Plot[Sin[x], {x, 0, 40Pi}, AspectRatio → 0.2]    (* пред. рисунок справа *)
```

Опция `AspectRatio→число` определяет отношение длин осей графика (высота/длина). В следующем примере физический наклон линии отличается от того, как мы себе это представляем из вида функции

```
Plot[2x, {x, 0, 5}]
```

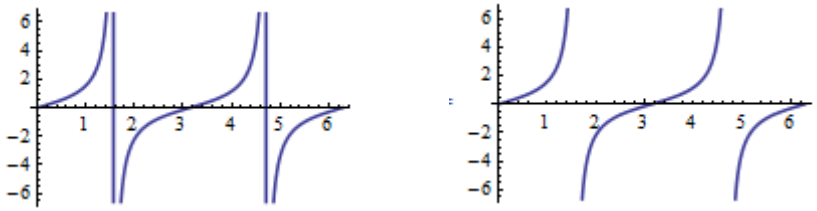
Так получается потому, что по умолчанию задается разный масштаб в горизонтальном и вертикальном направлении. Установка `AspectRatio→ Automatic` обеспечивает для этого примера одинаковый масштаб по *x* и *y*

```
Plot[2x, {x, 0, 5}, AspectRatio → Automatic]
```

Для исключения из графика особых точек используется опция `Exclusions`, которой передается список точек, в которых график не следует строить.

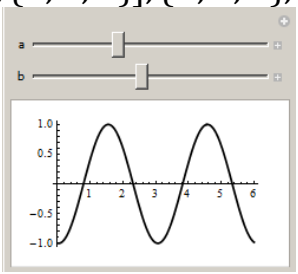
$$\mathbf{Plot[Tan[}x\mathbf{],\{}x\mathbf{,0,2Pi\}]}$$ (* следующий график слева *)

$$\mathbf{Plot[Tan[}x\mathbf{],\{}x\mathbf{,0,2Pi\},Exclusions\rightarrow\{}x=\mathbf{Pi/2,3Pi/2\}];}$$ (*след. гр. справа *)



Начиная с версии *Mathematica* 6 в системе появились специальные графические панели, содержащие элементы, с помощью которых можно динамически управлять выводом, в частности, строить графики. В следующем примере мы используем элемент «манипулятор»

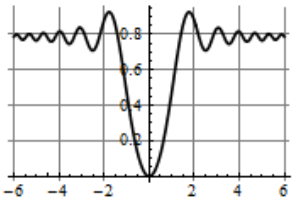
$$\mathbf{Manipulate[Plot[Sin[}a x+b\mathbf{],\{}x\mathbf{,0,6\}],\{}a\mathbf{,1,4\},\{}b\mathbf{,0,10\}]}$$



Перемещая указатели, в этом примере мы меняем значение параметров *a* и *b*, используемых при построении графика.

При построении графиков иногда нужно использовать функцию `Evaluate[expr]`. Она символьно вычисляет выражение, стоящее в ее аргументе. В следующем примере команда `Evaluate` заставляет ядро *Mathematica* сначала вычислить интеграл символьно, а потом построить график. Иначе для каждого конкретного *x* интеграл будет вычисляться заново.

$$\mathbf{Plot[Evaluate}\left[\int_0^x\int_0^z \mathbf{Cos[}zt\mathbf{]}\,dt\,dz\right]\mathbf{,\{}x\mathbf{,-6,6\},GridLines\rightarrow Automatic]}$$



Если попробовать построить график следующей командой, без использования функции `Evaluate`, то вероятно вам придется прервать вычисление с помощью комбинации клавиш `Alt` - **.**(точка).

$$\mathbf{Plot[}\int_0^x\int_0^z \mathbf{Cos[}zt\mathbf{]}\,dt\,dz\mathbf{,\{}x\mathbf{,-6,6\}]}$$ (* не делайте так *)

Функция `Evaluate` выполняет символьные вычисления!

Функцию `Evaluate` приходится использовать тогда, когда выражение, график которого следует построить, создается с атрибутом `HoldFirst`. Многие объекты/функции системы *Mathematica* имеют атрибуты, которые

управляют поведением объектов/функций. Узнать атрибуты выражения или функции можно, например, так

Attributes[Cos]

{Listable, NumericFunction, Protected}

Приведенные здесь атрибуты означают, что функция `Cos` в качестве аргумента может принимать списки (Listable); функция `Cos` будет возвращать результат в виде десятичного числа, если аргумент является десятичным числом (NumericFunction); и имя `Cos` защищено (Protected), т.е. если пользователь попытается создать переменную с именем `Cos`, то получит сообщение об ошибке.

Атрибут `HoldFirst` функции означает, что ее первый аргумент находится в невычисляемом состоянии. Например, наложим этот атрибут на функцию `Sin`.

SetAttributes[Sin, HoldFirst]

Attributes[Sin]

{HoldFirst, Listable, NumericFunction, Protected}

Теперь попытка построить график выражения `Sin[x]` не увенчается успехом.

Plot[Sin[x], {x, 0, 2π}] (* График не строится *)

Но, используя функцию `Evaluate`, вы можете все же его построить

Plot[Sin[Evaluate[x]], {x, 0, 2π}] (* Все Ок *)

После того, как мы «поиграли» с атрибутами функции `Sin`, снимем с нее атрибут `HoldFirst`.

ClearAttributes[Sin, HoldFirst]

Attributes[Sin]

{Listable, NumericFunction, Protected}

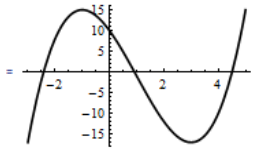
Еще один полезный случай использования функции `Evaluate` продемонстрирован в следующем примере.

$i = 0$;

f[x_] $= x^3 - 3x^2 - 9x + 10$;

g = Function[{x}, i + +; $f[x]$] (* создаем функцию g *)

Plot[$g[x]$, {x, −3, 5}]



i

453

После построения графика переменная i равна 453. Это значит, что определение функция g[x] создавалось 453 раза, потому, что при вычислении значения функции в каждой точке графика также создавалось определение функции! Но

$i = 0$; Plot[Evaluate[$g[x]$], {x, −3, 5}]; i

1

Переменная i=1. Значит определение функция g[x] создавалось только 1 раз. Затем, созданное определение, использовалось для вычисления значений в каждой точке графика. Второй подход значительно ускоряет вычисления. Такое поведение использовалось нами при построении графика функции $\int_0^x \int_0^z \mathrm{Cos}[zt]\, dt\, dz$. Снова выполним код

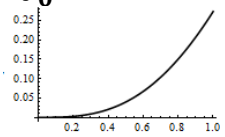
$\boldsymbol{i = 0;\ \mathbf{Plot}[g[x], \{x, -3, 5\}];\ i}$

451

Без использования функции `Evaluate` определение функции g[x] снова создавалось 451 раза.

Для определения времени вычисления в секундах какой – либо команды можно использовать функцию `Timing`.

$$\mathbf{Timing}[\mathbf{Plot}[\mathbf{Evaluate}[\int_0^x \mathbf{Sin}[t]^2\, dt], \{x, 0, 1\}]]$$

{0.046800, 

}

Без использования функции `Evaluate` время работы команды значительно больше (в различных версиях Mathematica это время будет колебаться от сотых долей секунды до десятков секунд; кроме того время, конечно, зависит от быстродействия компьютера)

$$\mathbf{Timing}[\mathbf{Plot}[\int_0^x \mathbf{Sin}[t]^2\, dt, \{x, 0, 1\}];]$$

$\{12.636081, \mathrm{Null}\}$

Еще с одним случаем, когда при построении графиков требуется использование функции `Evaluate`, мы столкнемся при решение дифференциальных уравнений.

2.1.2 Параметрические графики.

График кривых, задаваемых параметрически X=X(*t*), Y=Y(*t*), строится с помощью функции `ParametricPlot`.

$$\mathbf{ParametricPlot}[\{t^2, t\}, \{t, -2, 2\}]$$

Первый аргумент - список двух выражений {X(t),Y(t)}, которые определяют координаты *x* и *y* как функцию параметра *t*. Второй аргумент определяет диапазон изменения параметра *t*. Здесь *t* изменяется от -2 до 2.

Если вы хотите нарисовать более чем одну параметрическую функцию, используйте список списков

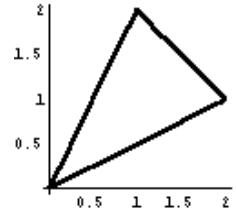
$$\mathbf{ParametricPlot}[\{\{t^3, t^2\}, \{t^2, t^3\}\}, \{t, -2, 2\}]$$

Большинство опций функции `Plot` могут использоваться также и в `ParametricPlot`. Например, опция `AspectRatio->Automatic` используется для одинакового масштабирования по осям. Опция `PlotStyle` может быть использована для рисования кривых с разной штриховкой, толщиной и цветом

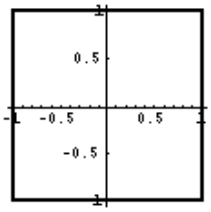
$$\mathbf{ParametricPlot}[\{\{t^3, t^2\}, \{t^2, t^3\}\}, \{t, -2, 2\},$$
$$\mathbf{PlotStyle} \to \{\{\}, \mathbf{Dashing}[\{0.02\}]\}, \mathbf{AspectRatio} \to \mathbf{Automatic}]$$

Эта функция позволяет нарисовать практически любую кривую, если вы можете составить ее параметрическое уравнение. Вот несколько примеров.

$$x[t\_] := \mathbf{Abs}[t] - 3\mathbf{Abs}[t-1]/2 + \mathbf{Abs}[t-3]/2;$$
$$y[t\_] := \mathbf{Abs}[t]/2 - 3\mathbf{Abs}[t-2]/2 + \mathbf{Abs}[t-3];$$
$$\mathbf{ParametricPlot}[\{x[t], y[t]\}, \{t, 0, 3\}, \mathbf{PlotStyle} \to \mathbf{Thickness}[0.01]]$$



$$\mathbf{xs}[t\_] := 1 - \mathbf{Abs}[t]/2 + \mathbf{Abs}[t-2]/2 + \mathbf{Abs}[t-4]/2 - \mathbf{Abs}[t-6]/2;$$
$$\mathbf{ys}[t\_] := 1 - \mathbf{Abs}[t-2]/2 + \mathbf{Abs}[t-4]/2 + \mathbf{Abs}[t-6]/2 - \mathbf{Abs}[t-8]/2;$$
$$\mathbf{ParametricPlot}[\{\mathbf{xs}[t], \mathbf{ys}[t]\}, \{t, 0, 8\}]$$

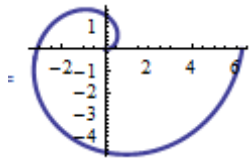


На сайте geometry@karazin.ua в нашем курсе «Аналитические методы геометрического моделирования» вы можете познакомиться с методикой составления параметрических и неявных уравнений составных кривых и поверхностей (уравнение кусков которых заданы). Эта методика научит вас составлять «аналитические» уравнения квадрата (см. выше), куба, пирамиды и практически любых непрерывных кривых и поверхностей.

2.1.3 График в полярных координатах

Для построения графиков в полярных координатах используется функция `PolarPlot` (в *Mathematica* 5 она находится в пакете расширения, загрузку которого надо выполнить командой `Needs["Graphics`Graphics`"]`). Начиная с 6 – й версии эта функция находится в ядре. Функция очень похожа на функцию `Plot`

$$\mathbf{PolarPlot}[t, \{t, 0, 2\mathbf{Pi}\}, \mathbf{PlotStyle} \to \mathbf{Thickness}[0.02]]$$



Первый аргумент – радиальная функция. Второй аргумент – интервал изменения угловой переменной *t*. `PolarPlot` автоматически выравнивает вертикальный и горизонтальный масштабы. В следующих примерах окружность похожа сама на себя, а не на овал

$$\mathbf{PolarPlot}[1, \{t, 0, 2\mathbf{Pi}\}]$$ (* центрированная окружность *)

или сдвинутая окружность

$$\mathbf{PolarPlot}[\mathbf{2Cos}[t],\{t,0,2\pi\}]$$

2.1.4 Графики неявных функций.

В *Mathematica* 5 для построения графиков кривых, определяемых неявными уравнениями, необходимо выполнить загрузку графического пакета `ImplicitPlot` командой `Needs["Graphics`ImplicitPlot`"]` или `<<Graphics`ImplicitPlot`.` После этого используется одноименная функция `ImplicitPlot`. Например

$$\mathbf{ImplicitPlot}[(x^2+y^2)^2==x^2-y^2,\{x,-1,1\},\{y,-0.5,0.5\}]$$

Для построения графиков кривых, заданных неявным уравнением, в *Mathematica* 6 и более поздних версиях используется функция `ContourPlot`. Эта функция очень похожа на `Plot`, только вместо явного выражения, содержащего одну переменную, вы задаете уравнение, содержащее две переменных

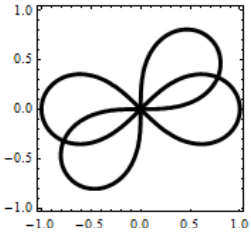
$$\mathbf{ContourPlot}[x^2+y^2==1,\{x,-1.5,1.5\},\{y,-1.5,1.5\}]$$

Первый аргумент – уравнение. В *Mathematica* уравнения обозначаются символом двойного равенства (==). Второй и третий аргументы задают диапазоны изменения горизонтальной и вертикальной переменных.

$$\mathbf{ContourPlot}[(x^2+y^2)^2==x^2-y^2,\{x,-1,1\},\{y,-0.5,0.5\},\ \mathbf{AspectRatio}\to\mathbf{Automatic}]$$

Можно рисовать несколько кривых на одном графике, задавая список уравнений в качестве первого параметра.

$$\mathbf{ContourPlot}[\{(x^2+y^2)^2==x^2-y^2,(x^2+y^2)^2==2xy\},\ \{x,-1,1\},\{y,-1,1\},\mathbf{ContourStyle}\to\mathbf{Thickness}[0.02]]$$

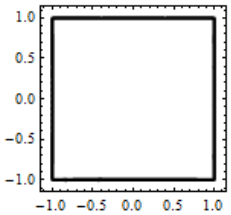


Обратите также внимание на то, что стиль кривых здесь задается опцией `ContourStyle`, а не `PlotStyle` как у других графических функций.

В следующем примере мы строим квадрат по его неявному уравнению.

$$W[x\_,y\_]=2-x^2-y^2-\sqrt{(1-x^2)^2+(1-y^2)^2}$$

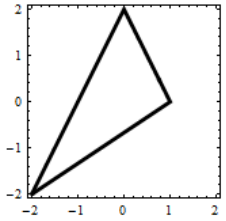
$$\mathbf{ContourPlot}[W[x,y]==0,\{x,-1.1,1.1\},\{y,-1.1,1.1\}]$$



Функция `ContourPlot` вычисляет значения выражения на некоторой регулярной сетке в плоскости XOY и рисует отрезки линии, когда разность

между двумя узлами меняет знак. Опция `PlotPoints->n` задает число точек разбиения интервалов изменения координат, определяя, тем самым, густоту сетки и, следовательно, гладкость кривой. Допустима форма опции `PlotPoints->{m,n}`, определяющей различное количество точек разбиения по осям.

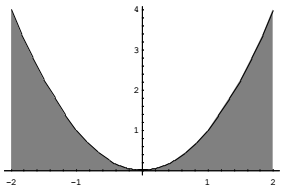
$$\mathbf{Wt[x\_, y\_] = 2 - x + y - Abs[x] - Abs[x - 2y - Abs[x]]}$$
$$\mathbf{ContourPlot[Wt[x, y] == 0, \{x, -2, 2\}, \{y, -2, 2\}, PlotPoints \to \{50, 50\}]}$$

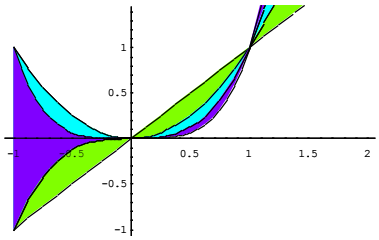


2.1.5 Области между кривыми.

В *Mathematica* 5 для построения графиков с заполненными областями между кривыми необходимо выполнить загрузку графического пакета `FilledPlot` командой `Needs["Graphics`FilledPlot`"]`и затем использовать одноименную функцию `FilledPlot`. Например

FilledPlot[x², {x,-2,2}] (* рисунок слева *)





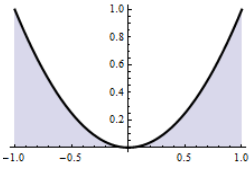
или

FilledPlot[{x, x², x³, x⁴}, {x,-1,2}] (* рисунок справа *)

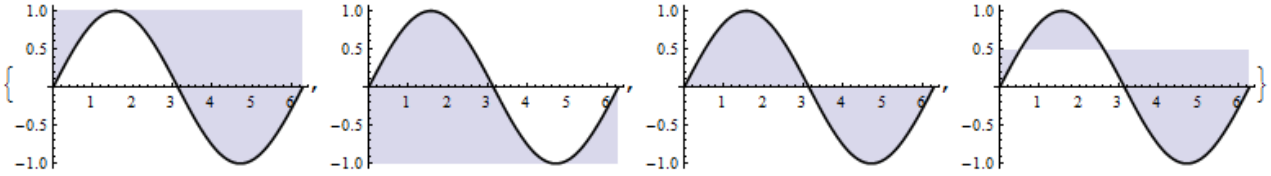
В *Mathematica* 6 и более поздних версиях для тех же целей используется опция `Filling` графических функций `Plot`, `ListPlot` и других.

С использованием этой опции можно задать уровни, которые определяют заполняемую область: между кривой и осью (Axis), между кривой и нижней границей графика (Bottom), между кривой и горизонтальной прямой на заданном уровне (число) и т.д.

$$\mathbf{Plot[x^2, \{x, -1, 1\}, Filling \to Axis]}$$

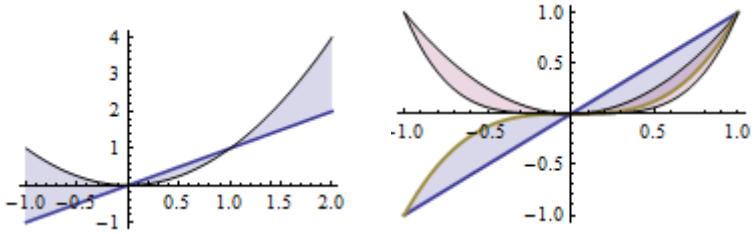


$$\mathbf{Table[Plot[Sin[x], \{x, 0, 2Pi\}, Filling \to f], \{f, \{Top, Bottom, Axis, 0.5\}\}]}$$



Эту опцию можно использовать для указания того, между какими кривыми графика следует заполнять область. Например, заполнение области между кривыми 1 и 2 выполняется командой

$\mathbf{Plot}[\{x, x^2\}, \{x, -1, 2\}, \mathbf{Filling} \to \{1 \to \{2\}\}]$ (* следующий рисунок слева *)

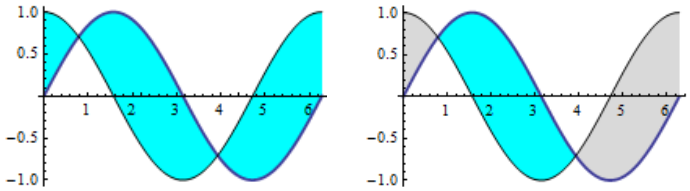


Вот пример заполнения областей между несколькими кривыми (предыдущий рисунок справа)

$\mathbf{Plot}[\{x, x^2, x^3, x^4\}, \{x, -1, 1\}, \mathbf{Filling} \to \{1 \to \{3\}, 2 \to \{4\}\}]$

Цвет заполнения определяется, например, так (следующий рисунок слева)

$\mathbf{Plot}[\{\mathbf{Sin}[x], \mathbf{Cos}[x]\}, \{x, 0, 2\mathbf{Pi}\}, \mathbf{Filling} \to \{1 \to \{\{2\}, \mathbf{Cyan}\}\}]$



Можно определить два цвета заполнения. В следующем примере, если первая кривая находится ниже второй, то используется первый цвет, иначе – второй.

$\mathbf{Plot}[\{\mathbf{Sin}[x], \mathbf{Cos}[x]\}, \{x, 0, 2\mathbf{Pi}\}, \mathbf{Filling} \to \{1 \to \{\{2\}, \{\mathbf{LightGray}, \mathbf{Cyan}\}\}\}]$

(предыдущий рисунок справа).

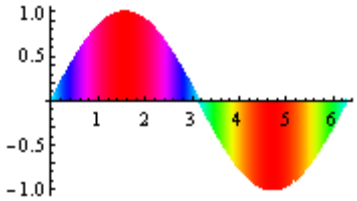
Опция `FillingStyle` используется для задания стиля заполнения областей. Предыдущие два графика можно построить с использованием этой опции так

$\mathbf{Plot}[\{\mathbf{Sin}[x], \mathbf{Cos}[x]\}, \{x, 0, 2\mathbf{Pi}\}, \mathbf{Filling} \to \{1 \to \{2\}\}, \mathbf{FillingStyle} \to \mathbf{Cyan}]$

$\mathbf{Plot}[\{\mathbf{Sin}[x], \mathbf{Cos}[x]\}, \{x, 0, 2\mathbf{Pi}\}, \mathbf{Filling} \to \{1 \to \{2\}\},$
$\mathbf{FillingStyle} \to \{\mathbf{LightGray}, \mathbf{Cyan}\}]$

Но опцию `FillingStyle` можно использовать и для определения более сложного стиля заполнения

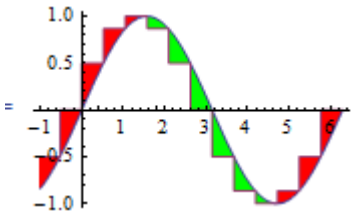
$\mathbf{Plot}[\mathbf{Sin}[x], \{x, 0, 2\mathbf{Pi}\}, \mathbf{ColorFunction} \to \mathbf{Function}[\{x, y\}, \mathbf{Hue}[y]],$
$\mathbf{Filling} \to \mathbf{Axis}, \mathbf{FillingStyle} -> \mathbf{Automatic}]$



Вот пример, в котором заполнение используется для обозначения области отличия функции от ее кусочно – постоянной интерполяции.

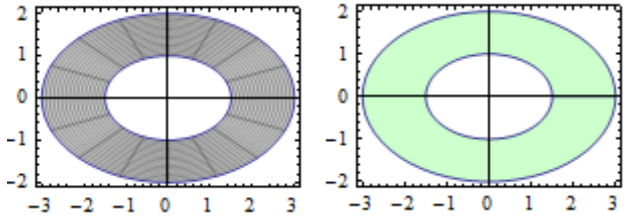
$\mathbf{si} = \mathbf{Interpolation}[\mathbf{Table}[\{x, \mathbf{Sin}[x]\}, \{x, -\pi/3, 2\pi, \pi/6\}],$
$\mathbf{InterpolationOrder} -> 0]$

$\mathbf{Plot}[\{\mathbf{Sin}[x], \mathbf{si}[x]\}, \{x, -\pi/3, 2\pi\}, \mathbf{Filling} \to \{1 \to \{2\}\},$
$\mathbf{FillingStyle} -> \{Red, Green\}]$



Функция `ParametricPlot` не имеет опции `Filling`, но она имеет возможность рисовать области, заданные параметрическими уравнениями $x = x(u,v),\ y = y(u,v)$

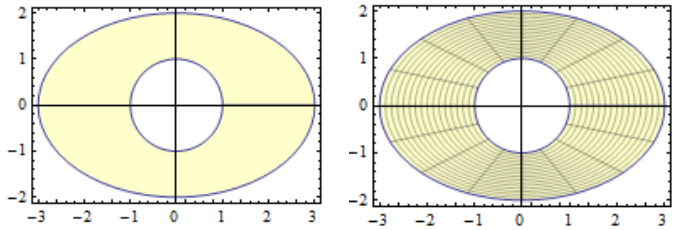
ParametricPlot[{{**3*v*Cos**[***u***], **2*v*Sin**[***u***]}}, {***u***, **0**, **2Pi**}, {***v***, **0.5**, **1**}] (* рис. слева *)
ParametricPlot[{{**3*v*Cos**[***u***], **2*v*Sin**[***u***]}}, {***u***, **0**, **2Pi**}, {***v***, **0.5**, **1**}, **Mesh** → **None**,
PlotStyle−> *Green*] (* рис. справа *)



Для заполнения области между кривыми $\mathbf{r_1}(u) = (x_1(u), y_1(u))\ (u_0 \le u \le u_1)$ и $\mathbf{r}_2(u) = (x_2(u), y_2(u))\ (u_0 \le u \le u_1)$, заданными параметрически, нужно записать параметрическое уравнение области между этими кривыми. Оно может иметь вид $\mathbf{r}(u,v) = v\,\mathbf{r}_1(u) + (1-v)\,\mathbf{r}_2(u),\ \ 0 \le v \le 1,\ (u_0 \le u \le u_1)$ или в покоординатной форме $x(u,v) = v\,x_1(u) + (1-v)x_2(u),\ \ y(u,v) = v\,y_1(u) + (1-v)y_2(u)$.

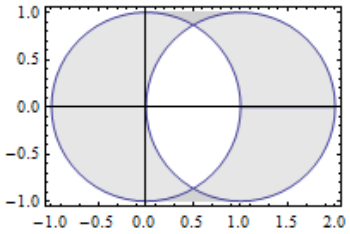
Заполним, например, область между единичной окружностью $\mathbf{r_1}(u) = (\cos(u), \sin(u))\ (0 \le u \le 2\pi)$ и эллипсом $\mathbf{r}_2(u) = (3\cos(u), 2\sin(u))$ $(0 \le u \le 2\pi)$. Имеем

r1[**u_**]: = {**Cos**[***u***], **Sin**[***u***]}
r2[**u_**]: = {**3Cos**[***u***], **2Sin**[***u***]}
ParametricPlot[{***v*r1**[***u***] + (**1** − ***v*)r2**[***u***]}, {***u***, **0**, **2*π***}, {***v***, **0**, **1**},
Mesh → **None**, **PlotStyle** → **Yellow**]



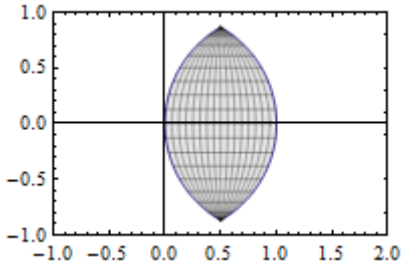
На правом графике мы построили ту же область, но оставили видимой координатную сетку (не использовали опцию `Mesh`). Из него видно, что заполняются точки области по прямым, соединяющим соответствующие (имеющие одинаковое значение параметра u) точки граничных кривых. Часто это не то, что вы имели ввиду. Например

r1[**u_**]: = {**Cos**[***u***], **Sin**[***u***]}
r2[**u_**]: = {**1** + **Cos**[***u***], **Sin**[***u***]}
ParametricPlot[{***v* r2**[***u***] + (**1** − ***v*)r1**[***u***]}, {***u***, **0**, **2*π***}, {***v***, **0**, **1**}, **Mesh** → **None**,
PlotStyle → **Gray**]



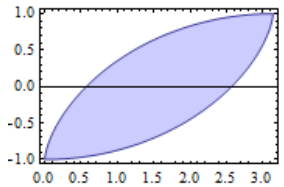
Чтобы заполнить луночку между окружностями надо записать параметрические уравнения дуг окружностей между точками их пересечения так, чтобы диапазоны изменения параметра u были одинаковыми и соответствовали желаемым направлениям движения вдоль дуг.

```
r1[u_]:={Cos[u],Sin[u]}
r2[u_]:={1-Cos[u],Sin[u]}
ParametricPlot[{v r2[u]+(1-v)r1[u]},{u,-Pi/3,Pi/3},{v,0,1},
                PlotStyle→Gray,PlotRange→{{-1,2},{-1,1}}]
```



Вот еще пример заполнения области между кривыми, заданными параметрически

```
r1[u_]:={u+Sin[u],-Cos[u]}
r2[u_]:={u-Sin[u],-Cos[u]}
ParametricPlot[{v r1[u]+(1-v) r2[u]},{u,0,Pi},{v,0,1},
                PlotStyle→Blue,Mesh→None]
```

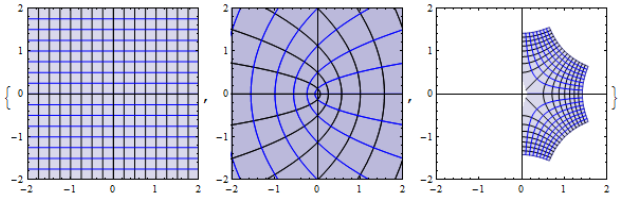


Пусть дана аналитическая функция $\mathbf{w}=f(\mathbf{z})$ комплексного переменного $\mathbf{z}=x+iy$ в плоскости Z и $\mathbf{w}=u+iv$ – комплексная переменная в плоскости W. Одним из способов ее графического представления является построение сеток кривых в плоскостях Z и W, соответствующих друг другу при таком отображении. Умение функции `ParametricPlot` рисовать области, заданные параметрическими уравнениями $x=x(u,v)$, $y=y(u,v)$, позволяет решать эту задачу.

В следующем примере мы строим декартову сеть в плоскости комплексного переменного Z и ее образы в плоскостях W, полученные при отображении $w=z^2$ и $w=\sqrt{z}$.

```
Table[
  ParametricPlot[{Re[(u+iv)^n],Im[(u+iv)^n]},{u,-2,2},{v,-2,2},
```

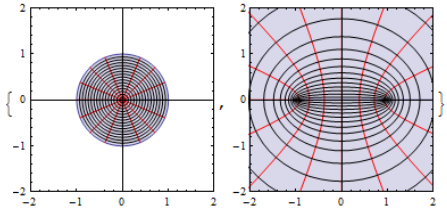
```
MeshStyle → {Black, Blue}, PlotRange → 2],
{n, {1, 2, 1/2}}]
```



Слева показана декартова сеть комплексной плоскости Z, в середине – ее образ в плоскости W при отображении $w = z^2$, справа – образ при отображении $w = \sqrt{z}$.

Рассмотрим функцию Жуковского $w = f(z) = \frac{1}{2}\left(z + \frac{1}{z}\right)$. Известно, что она реализует однолистное отображение, например, в области единичного круга $|z| < 1$. Построим картину отображения полярной системы координат плоскости Z в плоскость W.

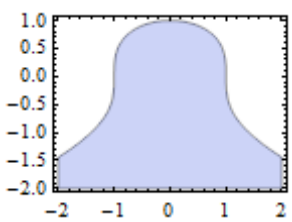
```
f[u_, r_] = 1/2 (r Cos[u] + i r Sin[u] + 1/(r Cos[u] + i r Sin[u]));
{ParametricPlot[{Re[r Cos[u] + i r Sin[u]], Im[r Cos[u] + i r Sin[u]]},
  {u, 0, 2π}, {r, 0, 1}, MeshStyle → {Red, Black}, PlotRange → 2],
 ParametricPlot[{Re[f[u, r]], Im[f[u, r]]}, {u, 0, 2π}, {r, 0, 1},
  MeshStyle → {Red, Black}, PlotRange → 2]}//Quiet
```



Здесь на левом рисунке показана сеть линий полярной системы координат в плоскости Z, на правом – образ кривых этой сети в плоскости W. Видно, что окружности переходят в эллипсы (черные кривые), прямые, проходящие через начало координат, переходят в гиперболы (красные кривые).

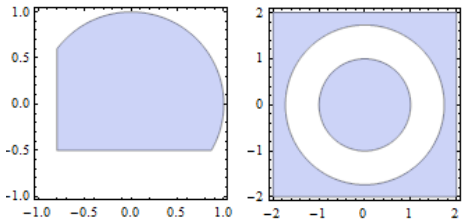
Для построения графиков заполненных областей предназначена функцию `RegionPlot`. Первым ее аргументом является *выражение*, зависящее, например, от переменных *x* и *y*, и которое может принимать значения `True` или `False`. Вторым и третьим аргументами являются диапазоны изменения переменных *x* и *y*, задаваемые в виде списков. Функция заполняет ту часть плоскости (*x*,*y*), в которой *выражение* принимает значение `True`.

```
RegionPlot[x^2 + y^3 < 1, {x, -2, 2}, {y, -2, 1}, AspectRatio → Automatic]
```



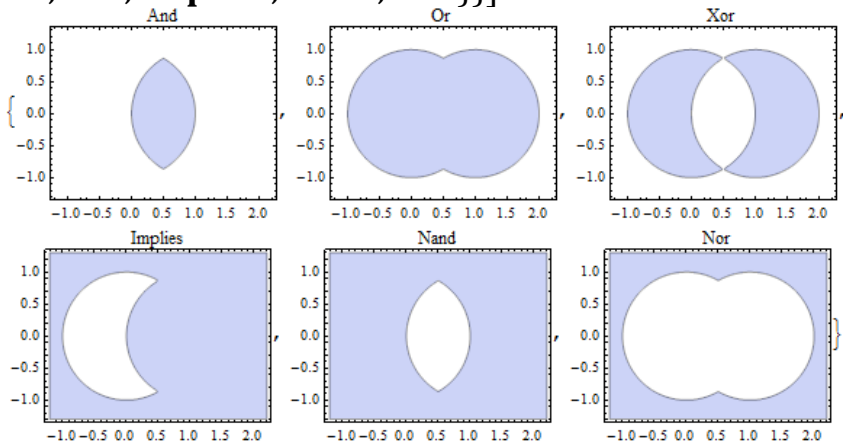
Выражения можно объединять с использованием логических операций `And(&&), Or(||), Xor, Implies, Nand, Nor` смысл которых ясен из их названия.

$$\mathbf{RegionPlot}[x^2+y^2<1\ \&\&\ y>-\frac{1}{2}\ \&\&\ x>-0.8,$$
$$\{x,-1,1\},\{y,-1,1\},\mathbf{AspectRatio}\to\mathbf{Automatic}]\quad (*\text{ след. рис. слева }*)$$
$$\mathbf{RegionPlot}[x^2+y^2<1\ ||\ x^2+y^2>3,\{x,-2,2\},\{y,-2,2\},$$
$$\mathbf{AspectRatio}\to\mathbf{Automatic}]\quad (*\text{ след. рис. справа }*)$$



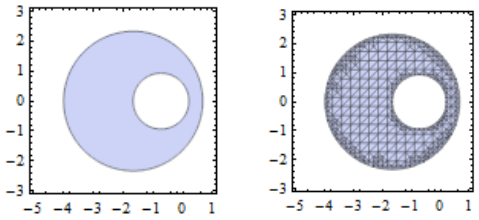
В следующем примере мы поясняем смысл логических операций между областями. Фактически здесь мы строим диаграммы Эйлера для операций над множествами.

$$a=x^2+y^2<1;$$
$$b=(x-1)^2+y^2<1;$$
$$\mathbf{Table}[$$
$$\quad\mathbf{RegionPlot}[f[a,b],\{x,-1.2,2.2\},\{y,-1.3,1.3\},$$
$$\quad\quad\mathbf{PlotLabel}\to f,\mathbf{AspectRatio}\to\mathbf{Automatic}],$$
$$\quad\{f,\{\mathbf{And},\mathbf{Or},\mathbf{Xor},\mathbf{Implies},\mathbf{Nand},\mathbf{Nor}\}\}]$$



Можно также использовать комплексные выражения для построения областей

$$\mathbf{RegionPlot}[1<Abs[\frac{(x+\mathbb{i}y)-3}{2(x+\mathbb{i}y)+1}]<2,\{x,-5,1\},\{y,-3,3\},$$
$$\mathbf{AspectRatio}\to\mathbf{Automatic}]$$



Правый рисунок построен предыдущей командой с добавлением опции `Mesh→All`. Обратите внимание, что в местах быстрого изменения функции, система автоматически сгущает сетку.

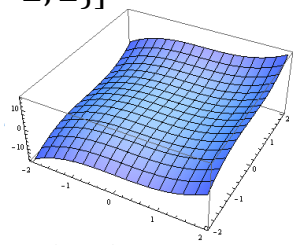
2.2 Трехмерные графики.

Система *Mathematica* снабжена большим количеством трехмерных графических функций. Основные из них описаны в данном разделе.

2.2.1 График функций заданных явно.

Для построения поверхности функции $z = f(x, y)$ предназначена функция `Plot3D`.

$\mathbf{Plot3D}[x^3 + y^3, \{x, -2, 2\}, \{y, -2, 2\}]$



Первый аргумент – выражение $f(x, y)$, график которого должен быть построен. Второй и третий аргументы определяют границы изменения переменных *x* и *y* , задаваемые в виде списков. Обычно функция вычисляется на сетке 15 на 15 и каждый кусок закрашивается согласно светоотражающей модели.

Mathematica размещает график в ящик, ориентация которого определяется положением точки, из которой он рассматривается. По умолчанию ориентация имеет следующие особенности:

- Шкала *x* почти горизонтальная, находится впереди слева, *x* увеличивается направо
- Шкала y проходит спереди назад вправо, *y* увеличивается назад.
- Шкала *z* - вертикальная слева, значения z возрастают вверх.

С помощью опций функции `Plot3D` ориентацию можно изменить. Для улучшения графика можно использовать опцию `Mesh→30` (в *Mathematica* 5 `PlotPoints->30)`, которая увеличивает количество точек до 30 (или любого другого числа).

$\mathbf{Plot3D}[\mathbf{Sin}[x\ \mathbf{Sin}[x\ y]], \{x, 0, 4\}, \{y, 0, 3\}, \mathbf{Mesh} \to \mathbf{30}]$

Размер сетки вырос с 15 на 15 по умолчанию до 30 на 30. Теперь посмотрим на график той же функции с другой точки зрения.

$\mathbf{Plot3D}[\mathbf{Sin}[x\mathbf{Sin}[xy]], \{x, 0, 4\}, \{y, 0, 3\}, \mathbf{Mesh} \to \mathbf{30}, \mathbf{ViewPoint} \to \{-2.4, 1., 2.\}]$

Здесь мы использовали опцию `ViewPoint`. Она определяет точку, из которой вы наблюдаете «куб», охватывающий поверхность.

По умолчанию *Mathematica* раскрашивает графики согласно простой модели освещенности. Цвет каждого участка зависит в основном от ориентации каждого сегмента поверхности относительно источников света.

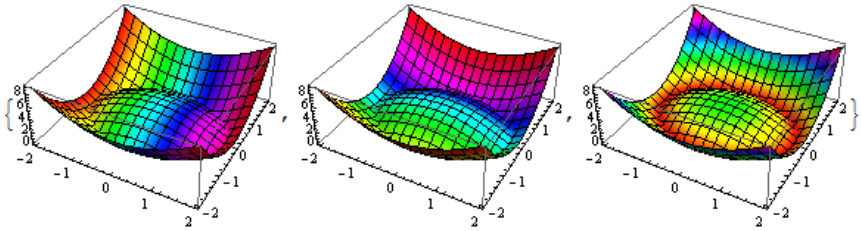
Такой способ раскраски иногда приводит к бедным цветным графикам. Если мы добавим опцию `Lighting->None`, то *Mathematica* отключит

модельное освещение (в *Mathematica* 5 `Lighting→False` и система будет закрашивать график, увеличивая яркость в соответствии со значением *z*).

$$\mathbf{Plot3D}[x^3 + y^3, \{x, -2, 2\}, \{y, -2, 2\}, \mathbf{Lighting} \to \mathbf{None}]$$

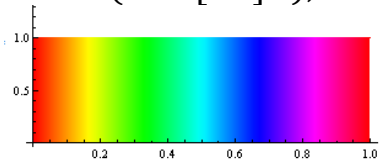
Управлять цветовой гаммой функции `Plot3D` можно с помощью опции `ColorFunction`. Она задается в виде функции и должна иметь определенное количество аргументов, которое зависит от графической функции, и должна возвращать строго определенный объект: `RGBColor`, `Hue` или `GrayLevel`. Для функции `Plot3D` функция `ColorFunction` будет принимать 3 аргумента – координаты *x*, *y*, используемые в выражении, и *z* – значение выражения. На поведение `ColorFunction` влияет опция `ColorFunctionScaling`, которая может принимать значения `True` или `False`. Если `ColorFunctionScaling→True` (значение по умолчанию), то все аргументы функции `ColorFunction` автоматически приводятся к диапазону [0, 1]. Если `ColorFunctionScaling→False` , то масштабирование аргументов не выполняется. Поскольку аргументы функций `RGBColor` и `Hue` должны изменяться в диапазоне от 0 до 1, то это в этом случае масштабирование аргументов становится вашей задачей.

$$\{\mathbf{Plot3D}[\mathbf{Abs}[3 - (x^2 + 2y^2)], \{x, -2, 2\}, \{y, -2, 2\}, \mathbf{ColorFunction} \to \mathbf{Function}[\{x, y, z\}, \mathbf{Hue}[x]]],$$
$$\mathbf{Plot3D}[\mathbf{Abs}[3 - (x^2 + 2y^2)], \{x, -2, 2\}, \{y, -2, 2\}, \mathbf{ColorFunction} \to \mathbf{Function}[\{x, y, z\}, \mathbf{Hue}[y]]],$$
$$\mathbf{Plot3D}[\mathbf{Abs}[3 - (x^2 + 2y^2)], \{x, -2, 2\}, \{y, -2, 2\}, \mathbf{ColorFunction} \to \mathbf{Function}[\{x, y, z\}, \mathbf{Hue}[z]]]\}$$



Посмотрите, как на графиках меняются цвета – в соответствии с изменением координат *x*, *y* или *z*. Здесь `Hue` – это графическая директива. В простейшей форме `Hue[h]` определяет, что используемый объект должен быть раскрашен тоном, задаваемым аргументом h $(0 \le h \le 1)$. На следующем рисунке вы можете увидеть значения цветового тона, который отсчитывается по горизонтальной оси, и сам цвет.

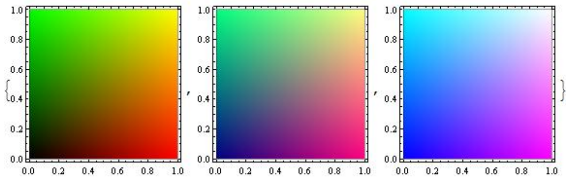
$$\mathbf{Plot}[\mathbf{1}, \{\mathbf{x}, \mathbf{0}, \mathbf{1}\}, \mathbf{ColorFunction} \to (\mathbf{Hue}[\#\mathbf{1}]\&), \mathbf{Filling} \to \mathbf{Axis}]$$



В приведенном здесь примере следует обратить внимание на то, что функция, передаваемая опции `ColorFunction` в функции `Plot`, принимает только два аргумента – координату *x* и значение выражения *y*.

Функция `RGBColor[r,g,b]` принимает три числовых аргумента в диапазоне от 0 до 1, каждый из которых определяет относительное содержание красной (r), зеленой (g) и синей (b) составляющих цвета. Она создает объект, который должна возвращать любая функция, представляющая значение опции `ColorFunction`. Вот пример цветовых палитр, создаваемых функцией `RGBColor[x,y,b]` при значении синей составляющей $b = \{0, 1/2, 1\}$.

$$\mathbf{Table}[\mathbf{RegionPlot}[\mathbf{True}, \{x, \mathbf{0}, \mathbf{1}\}, \{y, \mathbf{0}, \mathbf{1}\}, \mathbf{ColorFunction} \to \mathbf{Function}[\{x, y\}, \mathbf{RGBColor}[x, y, b]]], \{b, \{\mathbf{0}, \mathbf{1/2}, \mathbf{1}\}\}]$$



Обратите внимание на то, что функция, передаваемая опции `ColorFunction` в функции `RegionPlot`, принимает два аргумента – координаты x и y точек внутри области.

Напомним, что в *Mathematica* функции можно создавать по разному. Например, две следующие функции эквивалентны

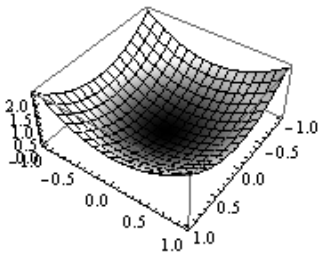
$\mathrm{RGBColor}[\#1, \#2, \#3]\&$

$\mathrm{Function}[\{x, y, z\}, \mathrm{RGBColor}[x, y, z]]$

#1 в первой записи (первый аргумент функции) соответствует x во второй записи. Аналогично, #2 и #3 в первой записи (второй и третий аргументы функции) соответствуют y и z во второй записи. Знак & (амперсанд) в первой записи является постфиксной формой функции `Function`. В примерах вы можете встретить обе эти формы записи функций.

Директива `GrayLevel[h]` определяет, что объект должен быть раскрашен серым цветом с интенсивностью h $(0 \le h \le 1)$. При этом 0 соответствует черному цвету, 1 – белому, промежуточные значения соответствуют серому цвету различной яркости.

$$\mathbf{Plot3D}[x^2 + y^2, \{x, -1, 1\}, \{y, -1, 1\}, \mathbf{ColorFunction} \to (\mathbf{GrayLevel}[\mathbf{2\#3}]\&)]$$



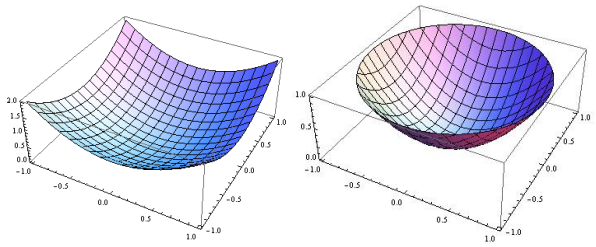
Подробнее о цветовых гаммах, используемых в системе, вы можете узнать, получив справку по этим функциями.

У многих графических функций есть еще одна важная опция `RegionFunction`. Она позволяет рисовать график поверхности $z = f(x, y)$ не во всем прямоугольнике, определяемом передаваемыми диапазонами $\{x, x_{\min}, x_{\max}\}$, $\{y, y_{\min}, y_{\max}\}$, а в некоторой его подобласти. Подобласть

определяется булевой функцией (для `Plot3D` – это функция трех переменных *x*, *y*, *z*). График рисуется только там, где эта булева функция принимает значение `True`. Сравните, например, два следующих графика, построенные функцией `Plot3D` без использования опции `RegionFunction` и с ее использованием.

$$\mathbf{Plot3D}[x^2+y^2,\{x,-1,1\},\{y,-1,1\}]$$

$$\mathbf{Plot3D}[x^2+y^2,\{x,-1,1\},\{y,-1,1\},\ \mathbf{RegionFunction}\to\mathbf{Function}[\{x,y,z\},x^2+y^2<1]]$$



Интересной является опция `MeshFunctions`. Она позволяет рисовать любую криволинейную сеть на поверхности. Для функции `Plot3D` эта опция может принимать список двух функций (не выражений!), например так

`MeshFunctions→{Функция1(x,y,z), Функция2(x,y,z)}`

Линии постоянного значения (линии уровня) `Функции1` определят одно семейство линий сети, линии уровня `Функции2` определят второе семейство. По умолчанию положение линий сети определяется равномерным распределением значений этих функций.

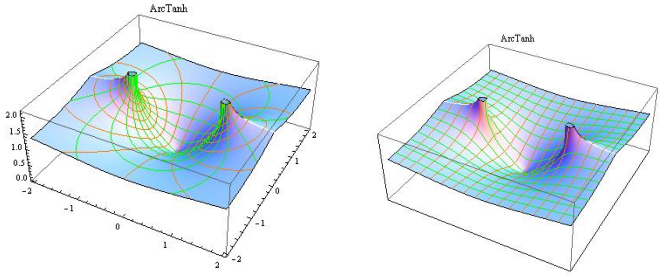
Если опция `MeshFunctions` не установлена, то используется значение по умолчанию `MeshFunctions→{#1&,#2&}`.

Вот пример одной и той же поверхности $z=\left|\operatorname{Arctanh}(x+i\,y)\right|$ с криволинейной и декартовой сетями на поверхности. В первом случае линии сети образуются линиями уровня вещественной и мнимой частей *z*.

$$f=\mathbf{ArcTanh};$$

$$\mathbf{Plot3D}[\mathbf{Abs}[f[x+Iy]],\{x,-2,2\},\{y,-2,2\},\mathbf{MeshFunctions}\to\{\mathbf{Function}[\{x,y,z\},\mathbf{Re}[f[x+Iy]]],\mathbf{Function}[\{x,y,z\},\mathbf{Im}[f[x+Iy]]]\},\ \mathbf{MeshStyle}\to\{\mathbf{Orange},\mathbf{Green}\},\mathbf{PlotLabel}\to f]$$

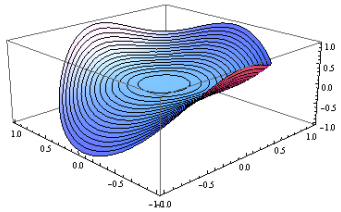
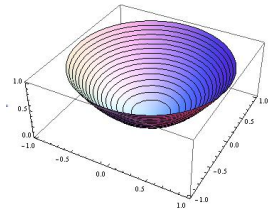
$$\mathbf{Plot3D}[\mathbf{Abs}[f[x+Iy]],\{x,-2,2\},\{y,-2,2\},\mathbf{MeshFunctions}\to\{\#1\&,\#2\&\},\ \mathbf{MeshStyle}\to\{\mathbf{Orange},\mathbf{Green}\},\mathbf{PlotLabel}\to f,\mathbf{Ticks}\to\mathbf{None}]$$



Конечно можно строить и одну сеть на поверхности (следующий рисунок слева)

$$\mathbf{Plot3D}[x^3+y^4,\{x,-1,1\},\{y,-1,1\},\ \mathbf{RegionFunction}\to\mathbf{Function}[\{x,y,z\},x^2+y^2<1],$$

$$\mathbf{MeshFunctions} \to \{\mathbf{Function}[\{x, y, z\}, x^2 + y^2]\}, \mathbf{Mesh} \to \mathbf{15}]$$

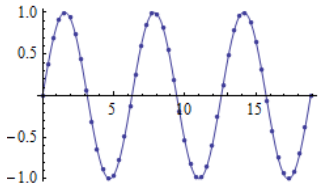
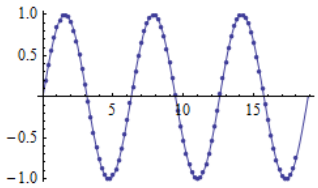
При задании функций для опций можно использовать z координату, чтобы не повторять выражение для функции

$\mathbf{Plot3D}[x^2 + y^2, \{x, -1, 1\}, \{y, -1, 1\},$
$\mathbf{RegionFunction} \to \mathbf{Function}[\{x, y, z\}, z < 1],$
$\mathbf{MeshFunctions} \to \{\mathbf{Function}[\{x, y, z\}, \mathbf{z}]\}, \mathit{Mesh} \to \mathbf{15}]$

(предыдущий рисунок справа).

Сетью для кривой является множество точек, расположенных на этой кривой. Например, для кривых, заданных явным уравнением, «сеть» можно построить так

$\mathbf{Plot}[\mathbf{Sin}[x], \{x, \mathbf{0}, \mathbf{6Pi}\}, \mathbf{Mesh} \to \mathbf{Full}]$ (* след. рисунок слева *)

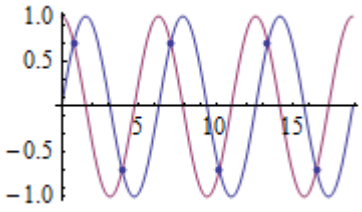
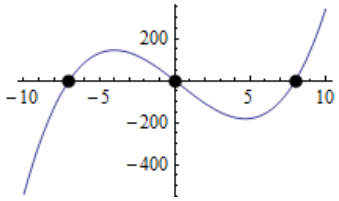



или можно задать список x – координат точек одномерной «сети»

$\mathbf{Plot}[\mathbf{Sin}[x], \{x, \mathbf{0}, \mathbf{6Pi}\}, \mathbf{Mesh} \to \{\mathbf{Table}[\mathbf{0.2}i, \{i, \mathbf{90}\}]\}]$ (* пред. рис. справа *)

Одномерную «сеть» можно использовать для обозначения точек пересечения кривых. Для этого достаточно задать Mesh – функцию, являющуюся разностью тех функций, пересечение которых нас интересует, и единственное нулевое значение для опции `Mesh`. В этом случае точками сети будут точки, для которых Mesh – функция обращается в ноль, т.е. точки пересечения кривых.

$\mathbf{Plot}[\{\mathbf{Sin}[x], \mathbf{Cos}[x]\}, \{x, \mathbf{0}, \mathbf{6Pi}\}, \mathbf{Mesh} \to \{\{\mathbf{0}.\}\},$
$\mathbf{MeshFunctions} \to \{\mathbf{Sin}[\#] - \mathbf{Cos}[\#]\&\}]$ (* след. рис. слева *)

Для обозначения точек пересечения кривой с осью X можно выполнить следующий код

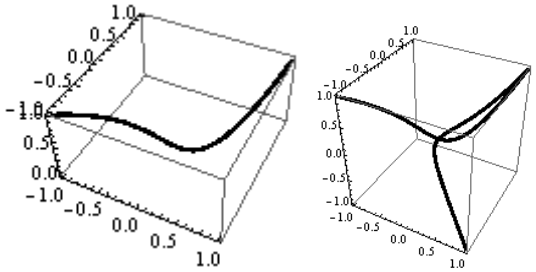
$f[\mathrm{x\_}] = x^3 - x^2 - 56x;$
$\mathbf{Plot}[f[x], \{x, -\mathbf{10}, \mathbf{10}\}, \mathbf{MeshFunctions} \to \{(\#\mathbf{2})\&\}, \mathbf{Mesh} \to \{\{\mathbf{0}\}\},$
$\mathbf{MeshStyle} \to \mathbf{PointSize}[\mathbf{Large}]]$ (* пред. рис. справа *)

Здесь #2 представляет значение функции.

2.2.2 График параметрических функций.

Существует две формы функции `ParametricPlot3D` для построения трехмерных кривых и трехмерных поверхностей. Для построения кривой необходимо выполнить команду

ParametricPlot3D$[\{t, t^3, t^2\}, \{t, -1, 1\}]$ (* следующий рисунок слева *)

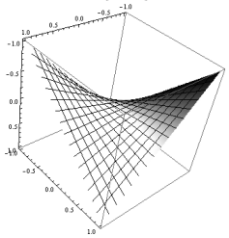


Первый аргумент – это список трех выражений, которые определяют *x*, *y* и *z* координаты, как функции параметра *t*. В нашем случае $x = t$, $y = t^3$, $z = t^2$. Второй аргумент/список определяет область изменения параметра (здесь от -1 до 1). Если вы хотите нарисовать более чем одну кривую на одном графике, вы должны задать список списков выражений, например

ParametricPlot3D$[\{\{t, t^3, t^2\}, \{t^2, t, t^3\}\}, \{t, -1, 1\}]$ (* пред. рисунок справа *)

Уравнение поверхности задается списком трех выражений с использованием двух параметров.

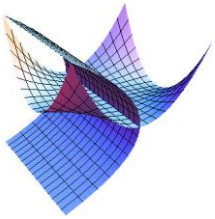
ParametricPlot3D$[\{uv, u, v\}, \{u, -1, 1\}, \{v, -1, 1\}]$



Первый аргумент – это список трех выражений, которые определяют *x*, *y* и *z* координаты, как функции двух параметров. В этом примере $x = uv$, $y = u$, $z = v$. Второй и третий аргументы – списки, которые определяют диапазоны изменения параметров *u* и *v*.

Большинство опций команды `Plot3D` можно использовать и с `ParametricPlot3D`. Например, следующие опции можно использовать для удаления границы ящика и осей (это полезно, когда поверхности пересекаются)

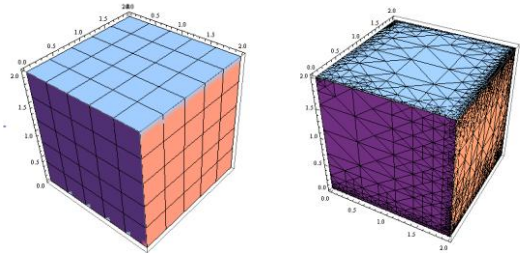
ParametricPlot3D$[\{\{u, -v^2, v\}, \{u, v, u^2v^2\}\}, \{u, -1, 1\}, \{v, -1, 1\},$
Boxed $\rightarrow$ **False**, **Axes** $\rightarrow$ **None**]



Для параметрических уравнений поверхностей с ребрами важно правильно использовать опцию `PlotPoints`. Она определяет начальное количество точек разбиения для диапазонов изменения параметров *u* и *v*. Вот пример

параметрических уравнений куба и его поверхности, построенной по таким уравнениям (следующий рисунок слева)

```
x[u_, v_] := Abs[u] - Abs[u - 1] - Abs[u - 2] + Abs[u - 3];
y[u_, v_] := 1 + Abs[v] - Abs[v - 1];
z[u_, v_] := 1 - Abs[Abs[u - 1] - Abs[u - 2] - Abs[u - 3] + Abs[u - 4] +
                    Abs[v - 1] - Abs[v - 2] + Abs[v] - Abs[v + 1]]/2 +
                    Abs[2 + Abs[u - 1] - Abs[u - 2] - Abs[u - 3] +
                    Abs[u - 4] + Abs[v - 1] - Abs[v - 2] + Abs[v] -
                    Abs[v + 1]]/2;
ParametricPlot3D[{x[u, v], y[u, v], z[u, v]}, {u, 0, 4}, {v, -1, 2},
                    PlotPoints → {21, 16}, Mesh → Full]
```



Чтобы ребра на графике не сглаживались нужно, чтобы через них проходили координатные линии. В нашем примере правильными значениями для опции `PlotPoints` будет пара чисел, определяемая по правилу $\{\Delta u \cdot n+1, \Delta v \cdot m+1\}$, где Δu - длина диапазона изменения параметра *u* (в нашем примере 4), Δv - длина диапазона изменения параметра *v* (в нашем примере 3), *m* и *n* любые натуральные числа. В примере мы взяли $n=5, m=5$. Поэтому значение обсуждаемой опции получилось следующим `PlotPoints→{21,16}`. Попробуйте в последней команде предыдущего кода задать другие значения опции, например, `PlotPoints→{22,17}`. Используемая здесь другая опция `Mesh→Full` определяет, что диапазоны изменения параметров разбиваются координатными линиями равномерно.

Другой способ правильно рисовать ребра состоит в сгущении координатной сетки в областях быстрого изменения координатных функций, т.е. в окрестности ребер. Эта процедура в функции `ParametricPlot3D` выполняется автоматически и максимальное количество делений ячеек координатной сетки определяется опцией `MaxRecursion`.

```
ParametricPlot3D[{x[u, v], y[u, v], z[u, v]}, {u, 0, 4}, {v, -1, 2},
                    Mesh → All, MaxRecursion → 4]
```

Приведенный здесь код строит поверхность куба, показанную на предыдущем рисунке справа. Как видим, сетка, по которой строилась поверхность, сгущается в окрестности ребер. Чтобы сетку отобразить использовалась опция `Mesh→All`.

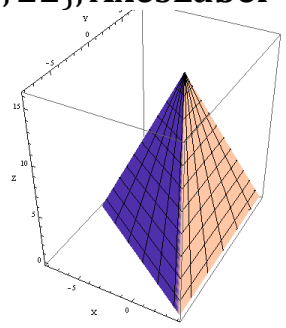
Если вы забыли, какие опции имеются у функции, то можно выполнить команду `Options`. В качестве аргумента ей надо передать имя функции, опции которой вы хотите узнать.

```
Options[ParametricPlot3D]
```

В окне документа будет отображен список всех опций функции и их значения.

Вот еще пример параметрических уравнений пирамиды.

```
x[u_, v_] := (2Abs[v] - 3Abs[v - 1] + Abs[v - 3]) *
        (-4 + 3Abs[u] - 3Abs[u - 1] - 3Abs[u - 2] + 3Abs[u - 3])/2;
y[u_, v_] := -(2Abs[v] - 3Abs[v - 1] + Abs[v - 3]) *
        (3Abs[u - 1] - 3Abs[u - 2] + Abs[u - 3] - Abs[u]);
z[u_, v_]: = 4(2 + Abs[v - 1] - Abs[v - 3]);
ParametricPlot3D[{x[u, v], y[u, v], z[u, v]}, {u, 0, 3}, {v, 0, 3},
        PlotPoints → {22, 22}, AxesLabel → {"X", "Y", "Z"}]
```



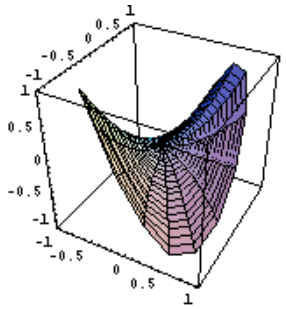
С методикой построения параметрических уравнений поверхностей с ребрами вы можете познакомиться по пособиям к нашему курсу «Аналитические методы геометрического моделирования», которые вы можете найти на сайте geometry@karazin.ua в разделе «Документы» автора.

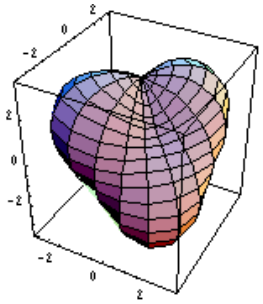
2.2.3 Графики в сферических и цилиндрических координатах.

В *Mathematica* 5 для построения графиков в сферических и цилиндрических координатах необходимо выполнить загрузку пакета `ParametricPlot3D` командой `Needs["Graphics`ParametricPlot3D`"]` и использовать функции `CylindricalPlot3D` и `SphericalPlot3D`. Например

```
CylindricalPlot3D[r² Sin[2 t],{r,0,1},{t,0,2π}]  (* рисунок слева *)
```

Первый аргумент задает *z* как функцию радиуса и полярного угла. Второй и третий аргументы определяют области изменения радиального параметра *r* и углового параметра *t*.





```
SphericalPlot3D[3+Sin[2 t]Sin[2 φ],{t,0, π},{ φ,0, 2π}]  (* рисунок справа *)
```

Первый аргумент задает радиус как функцию углов *t* и *φ*. Второй и третий аргументы определяют области изменения угла *t*, образуемого радиус – вектором точки с осью Z, и угла φ в плоскости XY.

В *Mathematica version* ≥ 6 для построения поверхности, уравнение которой задано в цилиндрических координатах (r, φ, z), используется функция `RevolutionPlot3D`. В этом случае формат ее вызова следующий

RevolutionPlot3D[f_z(r,φ),{r,r_{min},r_{max}},{φ,$φ_{min}$,$φ_{max}$}]

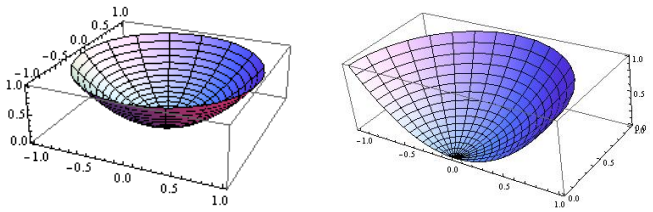
Вот пример построения предыдущей поверхности по ее уравнению в цилиндрической системе координат

RevolutionPlot3D[r^2**Sin**[2φ],{r,0,1},{φ,0,2π}] (* пред. рисунок слева *)

Первый аргумент представляет выражение вертикальной координаты $z = r^2 \sin(2\varphi)$ как функцию полярного радиуса r и угла φ. Второй и третий аргументы являются списками, определяющими интервалы изменения параметров r и φ. Полярный угол измеряется в радианах и отсчитывается против часовой стрелки от положительного направления оси X, если смотреть сверху.

Но функция RevolutionPlot3D имеет и другие возможности. В формате RevolutionPlot3D[f_z(x),{x,x_{min},x_{max}}] функция строит поверхность вращения кривой $z = f_z(x)$, заданной в плоскости XZ, вокруг оси Z.

RevolutionPlot3D[t^2,{t,0,1}] (* след. рисунок слева *)

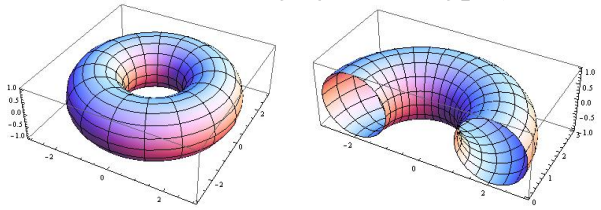


Можно построить часть поверхности вращения. Для этого нужно использовать диапазон изменения еще одной переменной, например θ, представляющую начальный и конечный углы поворота кривой относительно плоскости XZ.

RevolutionPlot3D[t^2,{t,0,1},{θ,0,π}] (* пред. рисунок справа *)

В формате RevolutionPlot3D[{f_x,f_z},{t,t_{min},t_{max}}] функция строит поверхность вращения кривой $x = f_x(t), z = f_z(t)$, заданной параметрически в плоскости XZ, вокруг оси Z.

RevolutionPlot3D[{2 + **Cos**[t],**Sin**[t]},{t,0,2**Pi**}] (* след. рисунок слева *)

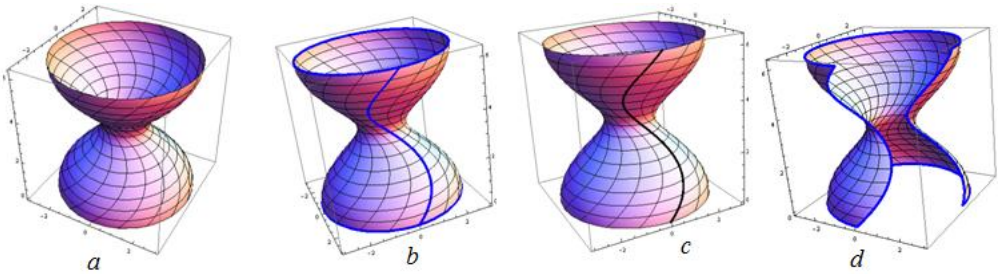


Аналогично предыдущему, можно построить часть поверхности вращения, задав диапазон угла поворота кривой относительно плоскости XZ.

RevolutionPlot3D[{2 + **Cos**[t],**Sin**[t]},{t,0,2π},{θ,0,π}] (*пред.рис.справа *)

В формате RevolutionPlot3D[{f_x,f_y,f_z},{t,t_{min},t_{max}}] строится поверхность вращения пространственной кривой $x = f_x(t), y = f_y(t), z = f_z(t)$, заданной параметрически, вокруг оси Z.

RevolutionPlot3D[{2 + **Cos**[t],**Sin**[t],t},{t,0,2**Pi**}] (* след. рисунок *a* *)



Чтобы выделить на поверхности ее края можно использовать опцию `BoundaryStyle`.

```
RevolutionPlot3D[{2 + Cos[t], Sin[t], t}, {t, 0, 2Pi},
    BoundaryStyle → Directive[Blue, Thick]]  (* пред. рисунок b *)
```

Краями поверхности считаются кривые, получаемые при граничных значениях независимых переменных. В случае поверхности вращения, кроме кривых, получаемых при вращении нижней и верхней точек (значения $t = 0$ и $t = 2\pi$ - верхняя и нижняя окружности), краями считаются начальное и конечное положение кривой вращения.

Опция `BoundaryStyle` имеется также у других графических функций, которые строят поверхности. Вместо значения `Thick` можно использовать графическую директиву `AbsoluteThickness[h]`, где `h` положительное число, указывающее толщину линии.

В последнем примере первым аргументом функции `RevolutionPlot3D` выступает параметрическое уравнение пространственной кривой. Ее можно изобразить отдельно функцией `ParametricPlot3D`. Чтобы объединить два графических объекта на одном графике (например, кривую в пространстве и поверхность) используется функция `Show`, которой через запятую передаются графические объекты. Следующий код строит кривую вращения и поверхность, и объединяет их на одном графике (предыдущий рисунок *c*).

```
Show[
    ParametricPlot3D[{2 + Cos[t], Sin[t], t}, {t, 0, 2Pi}],
    RevolutionPlot3D[{2 + Cos[t], Sin[t], t}, {t, 0, 2Pi}]]
```

Также как и выше, можно построить часть поверхности вращения. При этом мы показываем края поверхности (предыдущий рисунок *d*).

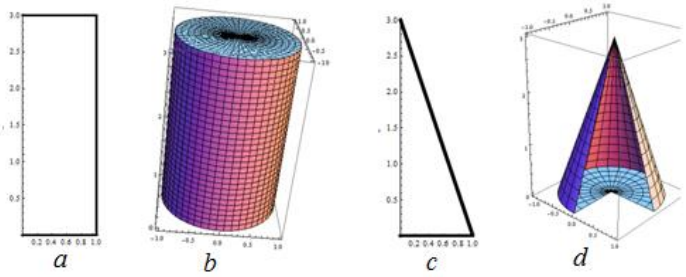
```
RevolutionPlot3D[{2 + Cos[t], Sin[t], t}, {t, 0, 2Pi}, {θ, 0, 3π/2},
    BoundaryStyle → Directive[Blue, AbsoluteThickness[4]]]
```

Заметим, что формат вызова `RevolutionPlot3D[`f_z`(x),{x,`x_{min}`,`x_{max}`}]` эквивалентен формату `RevolutionPlot3D[{t,0,`f_z`},{t,`t_{min}`,`t_{max}`}]`. А формат вызова `RevolutionPlot3D[{`f_x`,`f_z`},{t,`t_{min}`,`t_{max}`}]` эквивалентен формату `RevolutionPlot3D[{`f_x`,0,`f_z`},{t,`t_{min}`,`t_{max}`}]`.

Если в плоскости XZ кривая будет ломаной, то получаемая поверхность вращения будет иметь ребра.

На следующем рисунке (*a*) показана «скоба», параметрическое уравнение $x(t), y(t)$ которой приведено в следующем коде. Вращением этой ломаной получается поверхность цилиндра с донышками (рисунок *b*).

$x[t\_] = (5 - \mathbf{Abs}[t-1] - \mathbf{Abs}[t-4])/2;$
$y[t\_] = (3 + \mathbf{Abs}[t-1] - \mathbf{Abs}[t-4])/2;$
$\mathbf{ParametricPlot}[\{x[t], y[t]\}, \{t, 0, 5\}]$
$\mathbf{RevolutionPlot3D}[\{x[t], y[t]\}, \{t, 0, 5\}, \mathbf{PlotPoints} \to 51, \mathbf{Mesh} \to \mathbf{Full}]$



На предыдущем рисунке (c) показана ломаная, параметрическое уравнение $x(t), y(t)$ которой приведено в следующем коде. Вращением этой ломаной получается поверхность конуса с донышком (рисунок *d*).

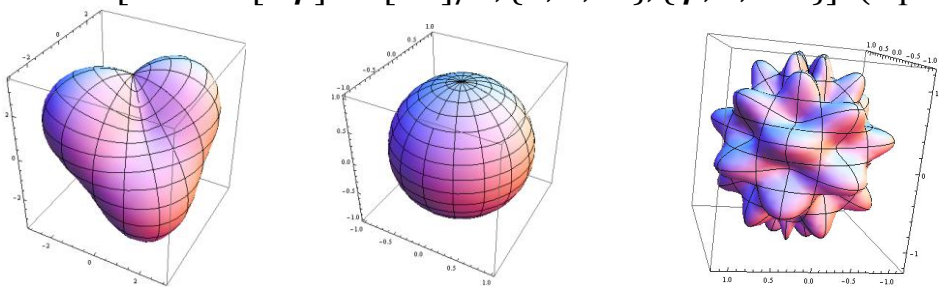
$$x[t\_] = \frac{t+2}{3} - \frac{2}{3}\mathbf{Abs}[t-1];$$
$x[t\_] = (t - 1 + \mathbf{Abs}[t-1])/2;$
$\mathbf{ParametricPlot}[\{x[t], y[t]\}, \{t, 0, 4\}]$
$\mathbf{RevolutionPlot3D}[\{x[t], y[t]\}, \{t, 0, 4\}, \{\theta, 0, 3\pi/2\},$
$\mathbf{PlotPoints} \to 21, \mathbf{Mesh} \to \mathbf{Full}]$

Функция `SphericalPlot3D` также как и `RevolutionPlot3D` теперь находится в ядре пакета.

$\mathbf{SphericalPlot3D}[3 + \mathbf{Sin}[2t]\mathbf{Sin}[2\varphi], \{t, 0, \pi\}, \{\varphi, 0, 2\pi\}]$ (* след. рис. слева *)

Первый аргумент представляет выражение длины радиус – вектора точки как функцию сферических углов *t* и *φ*. Второй и третий аргументы являются списками, которые задают интервал изменения угла *t*, образуемого радиус – вектором точки с осью Z, и интервал изменения угла φ, который образует проекция радиус – вектора на плоскость XY с фиксированным направлением в этой плоскости.

$\mathbf{SphericalPlot3D}[1, \{t, 0, \pi\}, \{\varphi, 0, 2\pi\}]$ (* следующий рисунок в центре *)
$\mathbf{SphericalPlot3D}[1 + \mathbf{Sin}[4\phi]\mathbf{Sin}[8\theta]/4, \{\theta, 0, \mathbf{Pi}\}, \{\phi, 0, 2\mathbf{Pi}\}]$ (* рис.справа *)

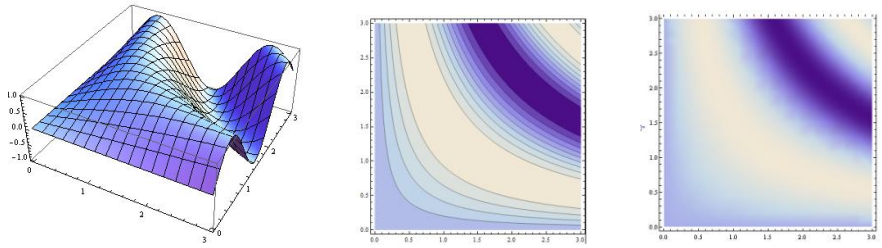


2.3 Другие графические возможности

2.3.1 Линии уровня функции

Функции `Plot3D`, `ContourPlot` и `DesityPlot` можно использовать почти взаимозаменяемо. Приведем примеры графиков при использовании всех трех функций:

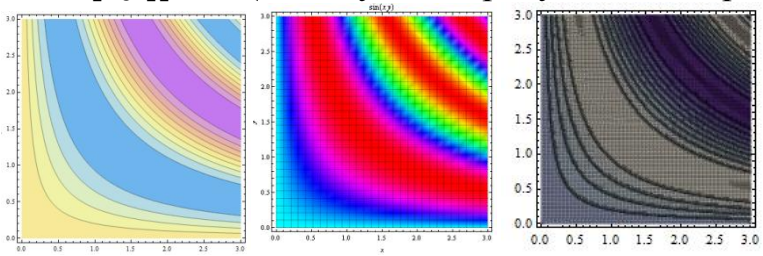
```
Plot3D[Sin[xy], {x, 0, 3}, {y, 0, 3}]          (* следующий рисунок слева *)
ContourPlot[Sin[xy], {x, 0, 3}, {y, 0, 3}]     (* следующий рисунок в центре *)
DensityPlot[Sin[xy], {x, 0, 3}, {y, 0, 3}]     (* следующий рисунок справа *)
```



В каждом случае первый аргумент задается как выражение для *z* координаты от аргументов *x*, *y*. Второй аргумент определяет область изменения *x*. Третий аргумент определяет область изменения *y*.

Функции `ContourPlot` и `DesityPlot` могут использовать большинство опций, которые есть у `Plot3D`. Например, можно увеличить количество точек, установить цвет и т.д.

```
ContourPlot[Sin[xy], {x, 0, 3}, {y, 0, 3}, PlotPoints → 20,
    ColorFunction → "Pastel"]                  (* следующий рисунок слева *)
DensityPlot[Sin[xy], {x, 0, 3}, {y, 0, 3}, PlotPoints → 30,
    Mesh → Full, ColorFunction → Hue, FrameLabel → {x, y},
    PlotLabel → Sin[xy]]       (* следующий рисунок в центре *)
```

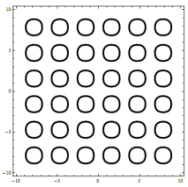


Одной из полезных опций для `DensityPlot` является `Mesh→False`. Это важно в том случае, когда строится высококачественная картинка с большим числом точек. Если оставить сетку, то можно получить очень черную картинку (предыдущий рисунок справа).

```
ContourPlot[Sin[xy], {x, 0, 3}, {y, 0, 3}, PlotPoints → 100, Mesh → All]
```

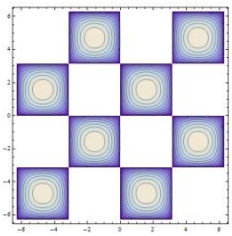
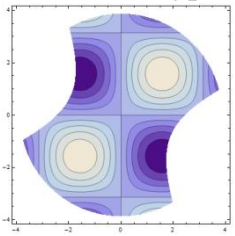
Напомним, что функцию `ContourPlot` мы ранее использовали для построения плоских кривых, уравнения которых заданы неявно.

```
ContourPlot[Sin[x]^2 Sin[y]^2 == 0.3, {x, -10, 10}, {y, -10, 10},
    ContourStyle → Thickness[0.01]]
```



Области изменения переменных, где решений нет (или решения комплексные) функциями `DensityPlot` и `ContourPlot` исключаются из построения автоматически (следующий рисунок слева)

$$\mathbf{ContourPlot}[\sqrt{\mathbf{Sin}[x]\mathbf{Sin}[y]}, \{x, -2\pi, 2\pi\}, \{y, -2\pi, 2\pi\}]$$

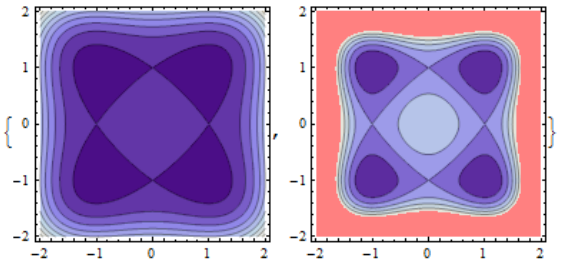
Но вы можете самостоятельно ограничить область отображения графиков с помощью опции `RegionFunction`. Она позволяет рисовать график не во всем прямоугольнике, определяемом передаваемыми диапазонами $\{x, x_{\min}, x_{\max}\}$, $\{y, y_{\min}, y_{\max}\}$, а в некоторой его подобласти. Подобласть определяется булевой функцией трех переменных *x*, *y*, *z*. График рисуется только там, где эта булева функция принимает значение `True`.

$$\mathbf{ContourPlot}[\mathbf{Sin}[x]\mathbf{Sin}[y], \{x, -4, 4\}, \{y, -4, 4\},$$
$$\mathbf{RegionFunction} \to \mathbf{Function}[\{x, y, z\}, x^2 - 2\,x\,y - y^2 < 6 \;\&\&\; x^2 + y^2 < 15]]$$

(предыдущий рисунок справа). Опция `RegionFunction` имеется у многих графических функций.

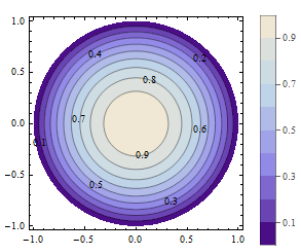
Когда мы строим контурный график выражения $f(x, y)$, то с помощью опции `PlotRange` можно уточнить диапазон изменения значений f, который следует обозначить на рисунке. При этом области плоскости XY, в которых значение f выходит за указанные в `PlotRange` пределы, обрезается. Опция `ClippingStyle` можно использовать для определения стиля/цвета заполнения обрезаемой области.

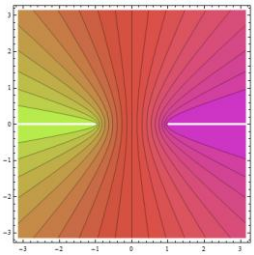
$$\{\mathbf{ContourPlot}[x^4 - 2x^2 + y^4 - 2y^2 + 1, \{x, -2, 2\}, \{y, -2, 2\},$$
$$\mathbf{ClippingStyle} \to \mathbf{None}],$$
$$\mathbf{ContourPlot}[x^4 - 2x^2 + y^4 - 2y^2 + 1, \{x, -2, 2\}, \{y, -2, 2\},$$
$$\mathbf{PlotRange} \to \{-2, 2\}, \mathbf{ClippingStyle} \to \mathbf{Pink}]\}$$



На линии уровня можно наносить метки с помощью опции `ContourLabels→True`, а опция `PlotLegends→Automatic` позволяет вывести рядом с графиком цветовую палитру.

```
ContourPlot[1 − x^2 − y^2, {x, −1, 1}, {y, −1, 1},
    ContourLabels → True, Contours → 9, PlotRange → {0, 1},
    ClippingStyle → None, PlotLegends → Automatic]
```





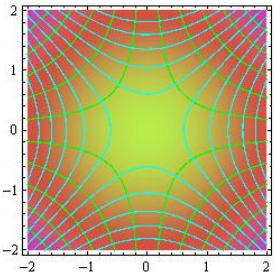
Использование комплексных функций также допустимо, но выражение, график которого строится, должно быть «вещественнозначным», а области «комплекснозначности» исключаются из построения.

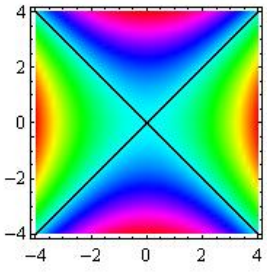
```
ContourPlot[Re[ArcSin[x + I y]], {x, −Pi, Pi}, {y, −Pi, Pi},
            ColorFunction → "NeonColors", Contours → 20]
```

(предыдущий график справа).

Также как и для других графических функций, в функциях `DensityPlot` и `ContourPlot` можно использовать опции `Mesh`, `MeshFunctions` и `MeshStyle` для построения криволинейной сети (следующий рисунок слева).

```
f = ((#1)^2)&;
DensityPlot[Abs[f[x + I y]], {x, −2, 2}, {y, −2, 2}, MeshFunctions →
    Function@@@{{{x, y, z}, Re[f[x + I y]]}, {{x, y, z}, Im[f[x + I y]]}},
    ColorFunction → NeonColors, Ticks → None, Mesh → 10,
    MeshStyle → {Cyan, Green}]
```





Опцию Mesh можно использовать для обозначения точек пересечения поверхности с плоскостью z=0

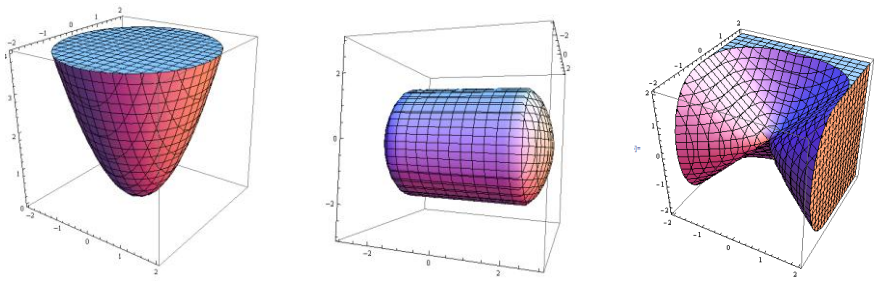
```
DensityPlot[y^2 − x^2, {x, −4, 4}, {y, −4, 4}, MeshFunctions → {#3&},
    MeshStyle → Thickness[Medium], Mesh → {{0}},
    ColorFunction → Hue, PlotPoints → 50]
```

(предыдущий рисунок справа). Здесь Mesh – функция возвращает значение $z = y^2 - x^2$, а линии сети строятся только для уровня z=0 (`Mesh→{{0}}`).

2.3.2 Трехмерные области и поверхности, заданные неявно

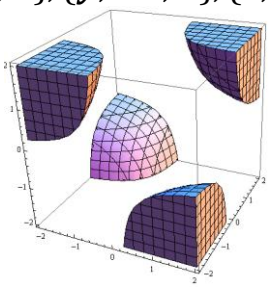
Функция `RegionPlot3D` рисует поверхность трехмерной области, используя булеву функцию (предикат) трех переменных.

```
RegionPlot3D[x^2 + y^2 − z < 0, {x, −2, 2}, {y, −2, 2}, {z, 0, 4}] (* рис. слева *)
```

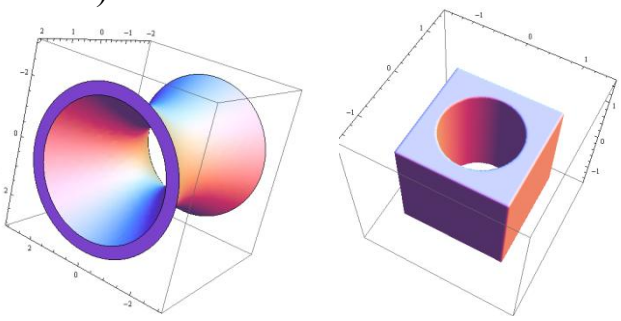


RegionPlot3D$[x^2+y^2+z^2 \le 9 \;\&\&\; y^2+z^2 \le 4,$
$\{x,-3,3\},\{y,-3,3\},\{z,-3,3\}]$
(предыдущий рисунок в центре).
RegionPlot3D$[x^2+y^3-z^2 > 0,\{x,-2,2\},\{y,-2,2\},\{z,-2,2\}]$
(предыдущий рисунок справа).

Область не обязана быть связной
RegionPlot3D$[xyz \ge 1,\{x,-2,2\},\{y,-2,2\},\{z,-2,2\}]$

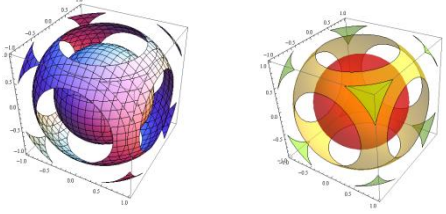


Предикат может быть составлен из любой логической комбинации неравенств
RegionPlot3D$[x^2+y^2-z^2 \le 3 \;\&\&\; x^2+y^2-z^2 \ge 1,$
$\{x,-3,3\},\{y,-3,3\},\{z,-2,2\},$ **Mesh** $\rightarrow$ **False**, **PlotPoints** $\rightarrow$ **30**
(следующий рисунок слева)



RegionPlot3D$[-1 \le x \le 1 \;\&\&\; -1 \le y \le 1 \;\&\&\; -1 \le z \le 1 \;\&\&$
$x^2+y^2 \ge \frac{1}{2},\{x,-1.5,1.5\},\{y,-1.5,1.5\},\{z,-1.5,1.5\},$
Mesh $\rightarrow$ **None**, **PlotPoints** $\rightarrow$ **100**] (* предыдущий рисунок справа *)

Функция `ContourPlot3D` предназначена для построения поверхностей постоянного значения (поверхностей уровня) функций трех переменных
ContourPlot3D$[x^2+y^2+z^2,\{x,-1,1\},\{y,-1,1\},\{z,-1,1\}]$ (* рис. слева *)



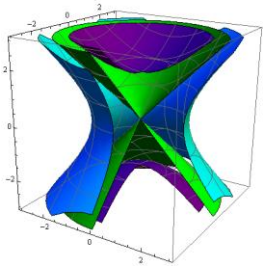
Поскольку поверхности часто закрывают друг друга, то их лучше рисовать полупрозрачными и не рисовать сетку.

```
ContourPlot3D[x^2 + y^2 + z^2, {x, -1, 1}, {y, -1, 1}, {z, -1, 1},
    ContourStyle → Opacity[0.5], Mesh → None,
    ColorFunction → (Hue[#4/4]&)]   (* предыдущий рисунок справа *)
```

Директива `Opacity[h]` определяет степень h прозрачности объекта $(0 \le h \le 1)$.

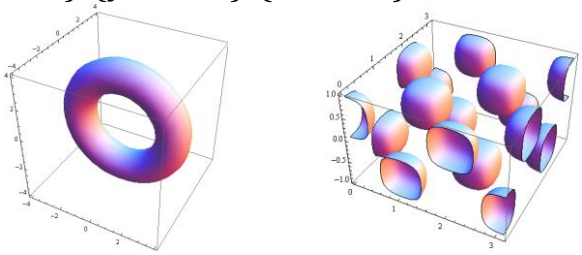
Можно также рисовать не всю поверхность, оставляя пространство для того, чтобы заглянуть внутрь поверхности.

```
ContourPlot3D[x^2 + y^2 - z^2, {x, -3, 3}, {y, -3, 3}, {z, -3, 3},
    Contours → {-3, 0, 3}, ContourStyle → {Purple, Green, Cyan},
    Mesh → 5, MeshStyle → GrayLevel[0.4],
    RegionFunction → Function[{x, y, z}, x < 0||y > 0]]
```



Функция `ContourPlot3D` может использоваться для построения поверхностей, уравнения которых заданы неявно. Вот пример неявного уравнения поверхности тора (следующий рисунок слева)

```
R = 3; r = 1;
ContourPlot3D[(x^2 + y^2 + z^2 + R^2 - r^2)^2 - 4R^2(x^2 + z^2) == 0,
    {x, -4, 4}, {y, -4, 4}, {z, -4, 4}, Mesh → None]
```



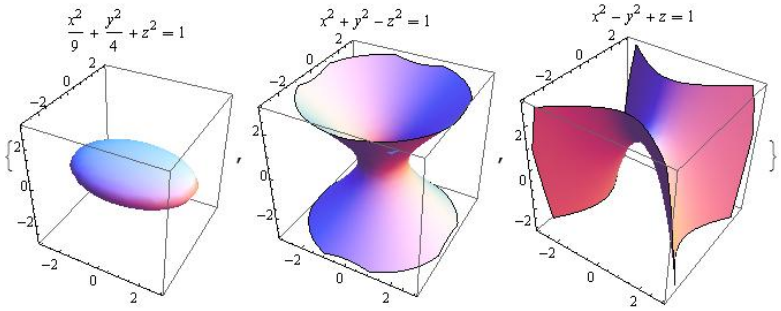
Неявное уравнение может задавать несвязную поверхность

```
ContourPlot3D[Cos[3x]Cos[3y]Sin[3z] == 1/4, {x, 0, π}, {y, 0, π}, {z, -1, 1},
    Mesh → None, BoxRatios → Automatic, PlotPoints → 10]
```

(предыдущий рисунок справа). Директива `BoxRatios` используется для задания пропорций между длинами сторон охватывающего параллелепипеда.

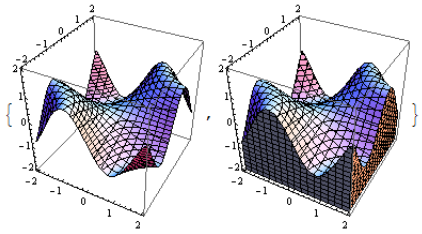
Вот пример построения поверхностей эллипсоида, однополостного гиперболоида и гиперболического параболоида по их неявным уравнениям.

```
Table[
  ContourPlot3D[Evaluate[ff], {x, -3, 3}, {y, -3, 3}, {z, -3, 3},
                PlotLabel → ff, Mesh → None],
  {ff, {x^2/9 + y^2/4 + z^2 == 1, x^2 + y^2 - z^2 == 1, x^2 - y^2 + z == 1}}]
```



Функции `ContourPlot3D` и `RegionPlot3D` могут строить одну и ту же явно заданную поверхность с той разницей, что во втором случае рисуются также боковые грани получаемого трехмерного тела.

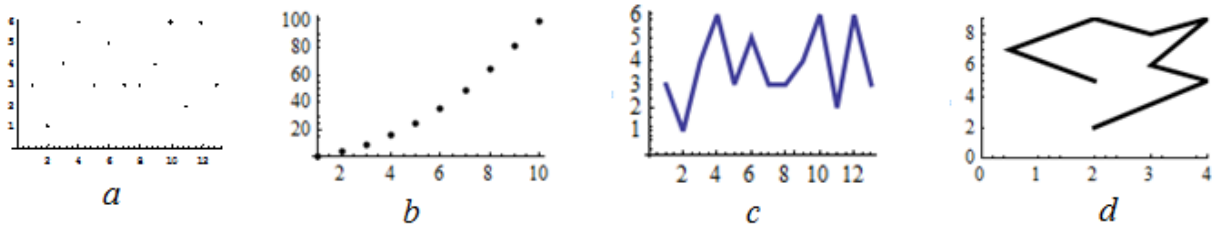
```
{ContourPlot3D[Sin[x y] == z, {x, -2, 2}, {y, -2, 2}, {z, -2, 2}],
 RegionPlot3D[Sin[x y] ≥ z, {x, -2, 2}, {y, -2, 2}, {z, -2, 2}]}
```



2.3.3 Графики списков значений.

Функция `ListPlot` графически изображает список значений в двух измерениях.

```
ListPlot[{3, 1, 4, 6, 3, 5, 3, 3, 4, 6, 2, 6, 3}]    (* следующий рис. a *)
```



Номер элемента в списке представляет горизонтальную координату точки, а само значение – вертикальную. Точки, которые используются для графика, часто бывают очень маленькими. Их можно увеличить с помощью опции `PlotStyle`

```
ListPlot[{3, 1, 4, 6, 3, 5, 3, 3, 4, 6, 2, 6, 3}, PlotStyle → PointSize[0.02]]
```

Число 0.02 в этом примере для опции `PointSize` определяет размер точки, как доля ширины графика.

Часто используют функцию `Table` вместе с функцией `ListPlot`

```
ListPlot[Table[k^2, {k, 10}], PlotStyle → {PointSize[0.03], Black},
PlotRange → All]    (* предыдущий рис. b *)
```

Чтобы точки соединить в *Mathematica* 5 использовалась опция `PlotJoined`.

```
ListPlot[{3, 1, 4, 6, 3, 5, 3, 3, 4, 6, 2, 6, 3}, PlotJoined → True]
```

Сейчас для соединения точек используется опция `Joined`.

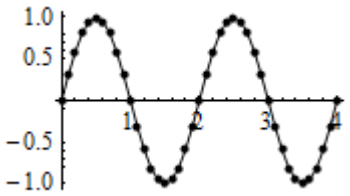
```
ListPlot[{3, 1, 4, 6, 3, 5, 3, 3, 4, 6, 2, 6, 3}, Joined → True,
                PlotStyle → Thickness[0.02]]    (* предыдущий рис. c *)
```

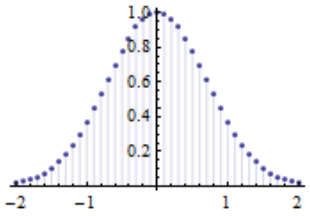
Если вы хотите определить значения x и y каждой точки, вы можете задать список пар чисел для `ListPlot`

$$\mathbf{ListPlot}[\{\{2,5\},\{0.5,7\},\{2,9\},\{3,8\},\{4,9\},\{3,6\},\{4,5\},\{2,2\}\},$$
$$\mathbf{Joined} \to \mathbf{True}, \mathbf{PlotRange} \to \{\{0,4\},\{0,9\}\},$$
$$\mathbf{PlotStyle} \to \{\mathbf{Thickness}[0.02], \mathbf{Black}\}] \quad (* \text{ предыдущий рис. } d \ *)$$

Как обычно существует очень много опций и дополнительных возможностей для `ListPlot`. Если вы хотите нарисовать кривую и точки, то можно использовать опцию `Mesh→Full` (следующий рисунок слева).

$$\mathbf{ListPlot}[\mathbf{Table}[\{x, \mathbf{Sin}[\pi x]\},\{x,0,4,0.1\}], \mathbf{Mesh} \to \mathbf{Full}, \mathbf{Joined} \to \mathbf{True},$$
$$\mathbf{PlotRange} \to \mathbf{All}, \mathbf{PlotStyle} \to \{\mathbf{PointSize}[0.02], \mathbf{Black}\}]$$



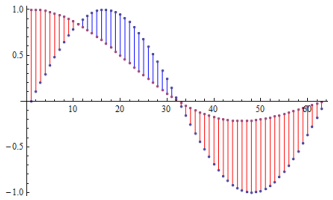


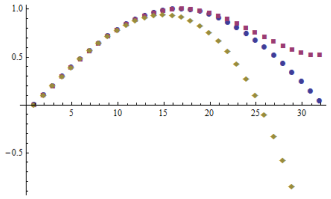
Как и у других графических функций, у `ListPlot` есть опция `Filling`.

$$\mathbf{ListPlot}[\mathbf{Table}[\{k, \mathbf{Exp}[-k^2]\},\{k,-2,2,0.1\}], \mathbf{Filling} -> Axis]$$

(предыдущий рисунок справа). Опцию `Filling`. можно использовать для выделения различий между множествами точек (следующий рисунок слева).

$$\mathbf{ListPlot}[\{\mathbf{Table}[\mathbf{Sin}[x],\{x,0,2\mathbf{Pi},0.1\}], \mathbf{Table}[\mathbf{Sinc}[x],\{x,0,2\mathbf{Pi},0.1\}]\},$$
$$\mathbf{Filling} \to \{1 \to \{2\}\}, \mathbf{FillingStyle} \to \{\mathbf{Red}, \mathbf{Blue}\}]$$





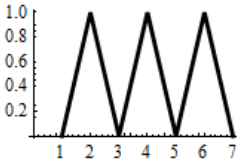
Можно использовать разные маркеры для обозначения точек.

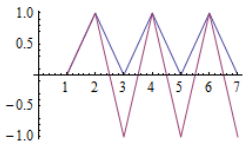
$$\mathbf{ListPlot}[\{\mathbf{Table}[\mathbf{Sin}[x],\{x,0,\pi,0.1\}], \mathbf{Table}[x - \frac{x^3}{3!} + \frac{x^5}{5!},\{x,0,\pi,0.1\}],$$
$$\mathbf{Table}[x - \frac{x^3}{3!},\{x,0,\pi,0.1\}]\}, \mathbf{PlotMarkers} \to \mathbf{Automatic}]$$

(предыдущий рисунок справа).

Таким образом, основное назначение функции `ListPlot` состоит в рисовании множества точек. Если вам нужно рисовать ломаную, проходящую через точки, то можно использовать функцию `ListLinePlot`. Если ее аргументом является одинарный список, то его элементы трактуются как вертикальные координаты точек, а горизонтальными являются номера точек в списке (следующий рисунок слева).

$$\mathbf{ListLinePlot}[\{0,1,0,1,0,1,0\}, \mathbf{PlotStyle} \to \{\mathbf{Thickness}[0.02], \mathbf{Black}\}]$$



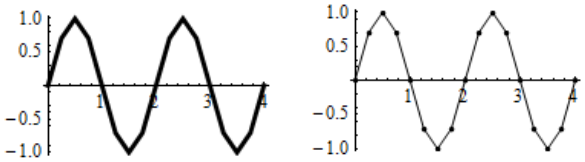


Можно строить ломаные для нескольких списков значений

ListLinePlot[{{**0**, **1**, **0**, **1**, **0**, **1**, **0**}, {**0**, **1**, −**1**, **1**, −**1**, **1**, −**1**}}]
(предыдущий рисунок справа).

Если вы хотите определить значения *x* и y каждой точки, вы можете задать список пар чисел для `ListLinePlot` (следующий рисунок слева)

ListLinePlot[**Table**[{***x***, **Sin**[***π x***]}, {***x***, **0**, **4**, **0. 25**}],
PlotStyle → {**Thickness**[**0. 02**], **Black**}]

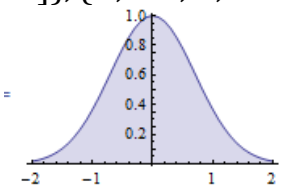


Используя опцию `Mesh`, вы можете нарисовать и ломанную и ее узлы

ListLinePlot[**Table**[{***x***, **Sin**[***π x***]}, {***x***, **0**, **4**, **0. 25**}],
PlotStyle → {**Black**}, **Mesh** → **All**]
(предыдущий рисунок справа).

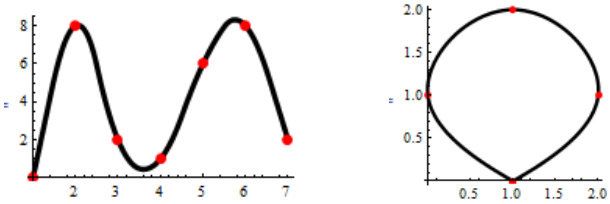
Как и других графических функций, у `ListLinePlot` есть опция `Filling`.

ListLinePlot[**Table**[{***k***, **Exp**[−***k*²]}, {***k***, −**2**, **2**, **0. 1**}], **Filling**−> *Axis*]



По умолчанию функция `ListLinePlot` строит интерполяционную ломаную, проходящую через множество точек. Но порядок интерполяционной кривой вы можете изменить. Использование опции `InterpolationOrder→n` означает, что кривая интерполируется кусочными полиномами степени n, которые сглаживаются между собой. Например, в следующем примере используется квадратичная сплайн – интерполяция.

ListLinePlot[{**0**, **8**, **2**, **1**, **6**, **8**, **2**}, **InterpolationOrder** → **2**,
Mesh → **Full**, **MeshStyle** → {**PointSize**[**0. 04**], **Red**},
PlotStyle → {**Thickness**[**0. 02**], **Directive**[**Black**]}]

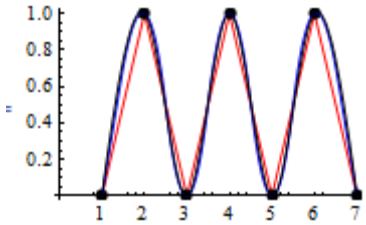


В следующем коде строится кубическая сплайн кривая

ListLinePlot[{{**1**, **0**}, {**2**, **1**}, {**1**, **2**}, {**0**, **1**}, {**1**, **0**}}, **InterpolationOrder** → **3**,
Mesh → **Full**, **MeshStyle** → {**PointSize**[**0. 04**], **Red**},
PlotStyle → {**Thickness**[**0. 02**], **Directive**[**Black**]},
AspectRatio → **Automatic**] (* предыдущий рисунок справа *)

Вот как выглядят интерполяционные кривые при разном порядке интерполирования и одном и том же множестве точек

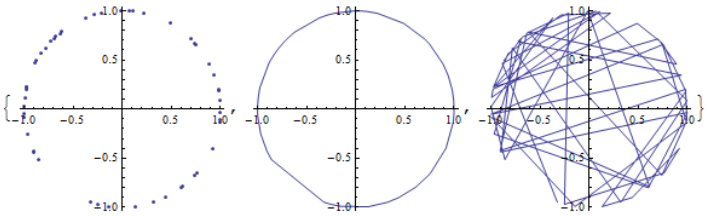
```
Show[
    ListLinePlot[{0, 1, 0, 1, 0, 1, 0}, Mesh → Full,
        MeshStyle → PointSize[0.04], PlotStyle → Red],
    ListLinePlot[{0, 1, 0, 1, 0, 1, 0},
        InterpolationOrder–> 2, PlotStyle → Blue],
    ListLinePlot[{0, 1, 0, 1, 0, 1, 0},
        InterpolationOrder–> 3, PlotStyle → Black]]
```



Напомним, что функция `Show` объединяет несколько графиков.

Функция `ListLinePlot` пытается нарисовать кривую, проходящую через множество точек в том порядке, в котором они заданы в списке, а функция `ListCurvePathPlot` пытается построить гладкую кривую, проходящую через множество точек без учета их последовательности в интерполируемом множестве. В следующем примере генерируются точки, лежащие на окружности, в случайном порядке.

```
data = Table[{Cos[t], Sin[t]}, {t, RandomReal[{0, 2Pi}, 50]}];
{ListPlot[data, AspectRatio → Automatic],
 ListCurvePathPlot[data],
 ListLinePlot[data, AspectRatio → Automatic]}
```



Здесь левый рисунок отображает множество точек, правый – ломаную, которая получена последовательным соединением точек в том порядке, в котором они идут в множестве/списке `data`, а средний – сглаженную функцией ListCurvePathPlot кривую.

Элементы вещественной матрицы, задаваемой в *Mathematica* как список списков, можно интерпретировать как яркости пикселей изображения. Функцией `Image` элементы такой матрицы, находящиеся в диапазоне от 0 до 1, рисуются в виде небольших квадратов; значение 0 соответствует черному цвету, значение 1 – белому, промежуточные значения соответствуют оттенкам серого цвета (следующий рисунок слева).

```
Image[{{0, 0.1, 0.2, 0.3}, {0.4, 0.5, 0.6, 0.7}, {0.8, 0.9, 1, 0}}]
```

Если элементы такой матрицы сами являются списками трех значений из диапазона от 0 до 1, то такие тройки/списки интерпретируются как цвет пикселя/квадрата в формате RGB.

$\mathbf{Image}[\{\{\{1,1,0\},\{1,0,1\},\{1,0,0\}\},$
$\{\{0,1,0\},\{0,0,1\},\{0,1,0\}\},$
$\{\{0.4,0.4,0.4\},\{1,0,1\},\{0,1,1\}\}\}]$ (* предыдущий рисунок в центре *)

Вот пример изображения матрицы случайных чисел

$n = 100; m = 80;$
$g = \mathbf{RandomReal}[\{0,1\},\{m,n\}];$
$\mathbf{Image}[g]$ (* предыдущий рисунок справа *)

Вот пример грубого представления круга как небольшой матрицы (следующий рисунок слева)

$f[\mathbf{i},\mathbf{j}] = \mathbf{If}[i^2 + j^2 \leq 4, 1, 0];$
$g = \mathbf{Table}[f[i,j],\{i,-3,3\},\{j,-3,3\}];$
$g//\mathbf{TableForm}$
$\mathbf{Image}[g]$

0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	1	1	0	0
0	1	1	1	1	1	0
0	0	1	1	1	0	0
0	0	0	1	0	0	0
0	0	0	0	0	0	0

Вот пример представления круга в виде матрицы большего размера

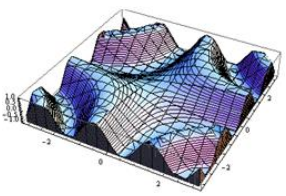
$R = 20; \mathbf{R1} = R + 5;$
$f[i,j] = \mathbf{If}[i^2 + j^2 \leq R^2, 1, 0];$
$g = \mathbf{Table}[f[i,j],\{i,-\mathbf{R1},\mathbf{R1}\},\{j,-\mathbf{R1},\mathbf{R1}\}];$
$\mathbf{Image}[g]$ (* предыдущий рисунок в центре *)

Вот пример генерирования случайного трехмерного массива и его отображения (предыдущий рисунок справа).

$\mathbf{Image}[\mathbf{RandomReal}[\{0,1\},\{12,12,3\}]]$

Кроме представления матрицы как изображения, функция `Image` умеет создавать растровое изображение из графического объекта. Например,

$\mathbf{pp} = \mathbf{RegionPlot3D}[\mathbf{Sin}[xy] > z,\{x,-\pi,\pi\},\{y,-\pi,\pi\},\{z,-1,1\},$
$\mathbf{BoxRatios} \to \{2\pi, 2\pi, 1\}];$
$\mathbf{pi} = \mathbf{Image}[\mathbf{pp}]$



Объект **pi** невозможно вращать, как обычный графический 3D объект – это уже растровое изображение. Но его можно сохранить в графический файл, с помощью функции `Export`.

```
Export["Surf01.gif", pi];
```

На моем компьютере файл сохраняется в каталог C:\Users\User\Documents, однако, это зависит от настроек в вашей системе.

Функция `Export` поддерживает несколько графических форматов с которыми вы можете познакомиться по справочной системе. Например

```
Export["Surf01.bmp", pi];
```

Прочитать растровое изображение можно с помощью функции `Import`.

```
pi2 = Import["Surf01.gif"]
```

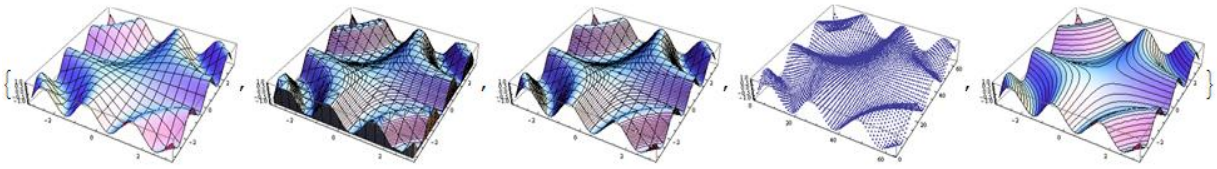
В результате будет показана картинка, приведенная на предыдущем рисунке.

GIF файлы способны хранить несколько растровых изображений, которые могут воспроизводиться как анимация. Создадим, например, список нескольких графических объектов (графиков)

```
p1 = Plot3D[Sin[xy], {x, -π, π}, {y, -π, π}, BoxRatios → {2π, 2π, 1}];
p2 = RegionPlot3D[Sin[xy] > z, {x, -π, π}, {y, -π, π}, {z, -1, 1},
                  BoxRatios → {2π, 2π, 1}];
p3 = ContourPlot3D[Sin[xy] - z == 0, {x, -π, π}, {y, -π, π}, {z, -1, 1},
                  BoxRatios → {2π, 2π, 1}];
p4 = ListPointPlot3D[Table[Sin[i * j], {i, -π, π, 0.1}, {j, -π, π, 0.1}],
                  BoxRatios → {2π, 2π, 1}];
p5 = Plot3D[Sin[xy], {x, -π, π}, {y, -π, π}, BoxRatios → {2π, 2π, 1},
                  MeshFunctions → {#1 * #2&}, Mesh → 50];
tp = {p1, p2, p3, p4, p5};
```

Преобразуем каждый из них в растровый образ и создадим из них список

```
rtp = Map[Image, tp]
```



Здесь функция `Map` применяет функцию `Image` (первый аргумент) к каждому элементу списка **tp** (второй аргумент).

Экспортируем список **rtp** в графический файл

```
Export["Surf02.gif", rtp];
```

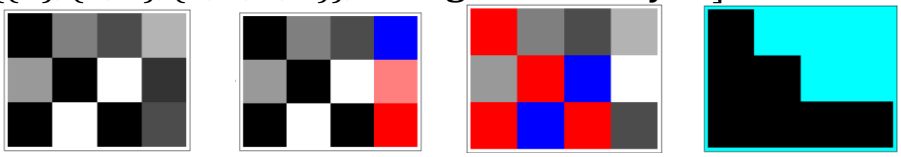
Теперь файл Surf02.gif содержит список растровых изображений, которые будут воспроизводиться как анимация.

Функция `Export` поддерживает несколько форматов анимированных файлов, с которыми вы можете познакомиться по справочной системе. Например, анимацию можно сохранить в файл формата AVI.

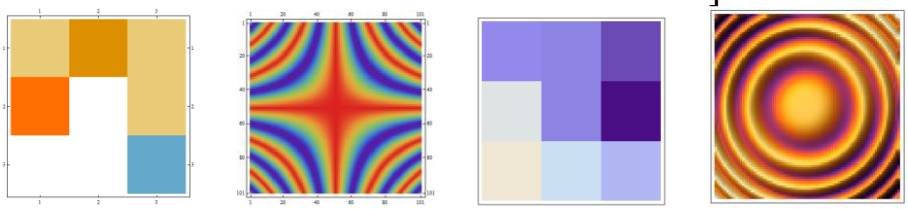
```
tl = Table[Plot[Sin[nx], {x, 0, 10}], {n, 1, 10, 0.2}];
rtl = Map[Image, tl];
Export["Surf03.avi", rtl];
```

Для отображения массивов данных предназначены функции `ArrayPlot`, `MatrixPlot` и `ReliefPlot`. Вот несколько примеров.

```
ArrayPlot[{{1, 0.5, 0.7, 0.3}, {0.4, 1, 0, 0.8}, {1, 0, 1, 0.7}}]
ArrayPlot[{{1, 0.5, 0.7, Blue}, {0.4, 1, 0, Pink}, {1, 0, 1, Red}}]
ArrayPlot[{{1, 0.5, 0.7, 0.3}, {0.4, 1, 00.3}, {1, 0, 1, 0.7}},
                        ColorRules -> {1 → Red, 0 → Blue}]
ArrayPlot[{{1}, {1, 1}, {1, 1, 1, 1}}, Background → Cyan]
```



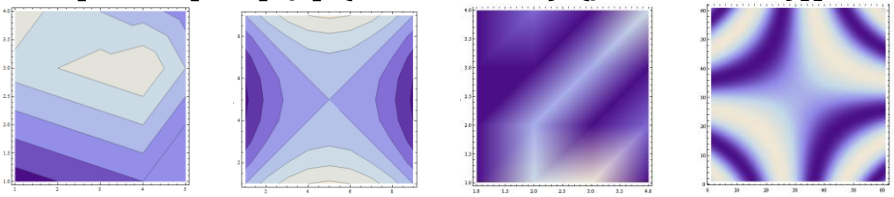
```
MatrixPlot[{{1, 2, 1}, {3, 0, 1}, {0, 0, -1}}]
MatrixPlot[Table[Cos[x y/150.], {x, -50, 50}, {y, -50, 50}],
                        ColorFunction → "Rainbow"]
ReliefPlot[{{4, 2, 1}, {3, 0, -2}, {0, 0, -1}}]
ReliefPlot[Table[i j/16 + Cos[i^2 + j^2], {i, -4, 4, .1}, {j, -4, 4, .1}],
                        ColorFunction → "SunsetColors"]
```



У функции `Image`, `ArrayPlot`, `MatrixPlot` и `ReliefPlot` есть много вариантов использования и опций с которыми вы можете познакомиться по справочной системе. В основном они используются при обработки изображений. Это отдельная большая тема, которая выходит за рамки нашего пособия.

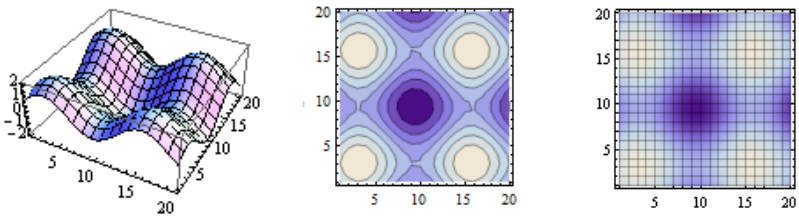
Функция `ListContourPlot` создает контурный график из двумерного массива, а функция `ListDensityPlot` создает график плотности.

```
ListContourPlot[{{1, 2, 3, 4, 6}, {5, 6, 7, 8, 4}, {9, 10, 11, 12, 8}, {12, 9, 8, 7, 5}}]
ListContourPlot[Table[i^2 - j^2, {i, -4, 4}, {j, -4, 4}]]
ListDensityPlot[{{1, 4, 5, 1}, {1, 3, 1, 2}, {1, 1, 3, 1}, {1, 2, 1, 4}}]
ListDensityPlot[Table[Sin[xy], {x, -3, 3, 0.1}, {y, -3, 3, 0.1}]]
```



Вот пример представления одного и того же набора данных с использованием трех разных функций

```
data = Table[Sin[i/2] + Sin[j/2], {i, 1, 20}, {j, 1, 20}];
ListPlot3D[data]
ListContourPlot[data]
ListDensityPlot[data, Mesh → Full]
```

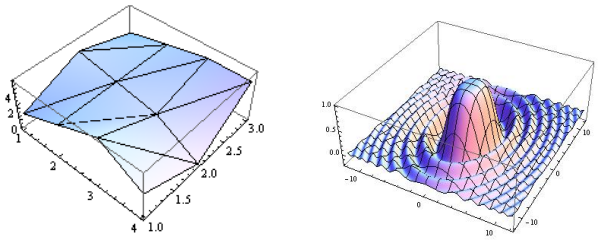


С другими вариантами использования этих функций вы можете познакомиться по справочной системе.

У многих двумерных функций, отображающих списки, есть свои трехмерные аналоги. Перечислим некоторые из них: `ListPlot3D`, `ListPointPlot3D`, `ListSurfacePlot3D`, `ListContourPlot3D`.

Функция `ListPlot3D` требует матрицу (список списков) значений z, заданных в узлах квадратной целочисленной сетки (i, j) на плоскости XY.

$$\mathbf{ListPlot3D}[\{\{1,3,5,2\},\{4,3,2,-1\},\{2,3,4,4\}\},\mathbf{Mesh}\to\mathbf{All}]$$ (* рис.слева *)



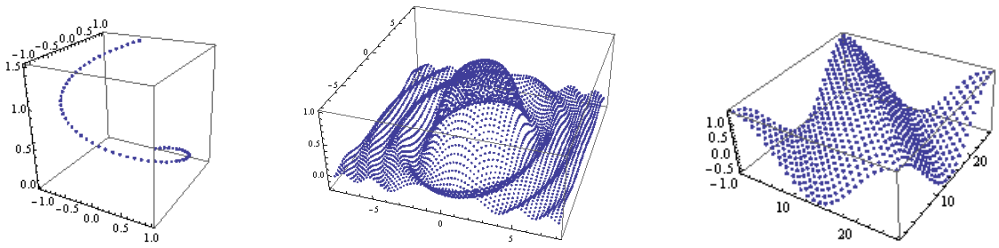
Первый аргумент должен представлять прямоугольный массив или он должен быть списком троек чисел $\{x_i, y_i, z_i\}$, где z_i представляет высоту поверхности в точке $\{x_i, y_i\}$. Вот пример второго формата вызова этой функции

$$\mathbf{data}=\mathbf{Flatten}[\mathbf{Table}[\{x,y,\mathbf{Sin}\left[\frac{x^2}{4}+\frac{y^2}{9}\right]/\left(\frac{x^2}{4}+\frac{y^2}{9}\right)\},$$
$$\{x,-12,12,0.2\},\{y,-12,12,0.2\}],1];//\mathbf{Quiet}$$
$$\mathbf{ListPlot3D}[\mathbf{data},\mathbf{PlotRange}\to\mathbf{All}]$$

(предыдущий рисунок справа). Здесь функция `Flatten[expr,1]` потребовалась для реорганизации тройного списка в двойной, который требует функция `ListPlot3D` во втором формате вызова. Кроме того, мы использовали постфиксный вызов функции `Quiet` для того, чтобы отменить вывод сообщения о делении на ноль в начале координат.

Приведем примеры использования других трехмерных «списочных» функций. Функция `ListPointPlot` принимает список пространственных координат точек $\{x_i, y_i, z_i\}$ и рисует их. Точки могут находиться как на кривой так и на поверхности. Вот пример множества точек, расположенных на кривой (следующий рисунок слева)

$$\mathbf{data}=\mathbf{Table}\left[\left\{\mathbf{Sin}[u],\mathbf{Cos}[u],\frac{u}{4}\right\},\{u,0,6,0.1\}\right];$$
$$\mathbf{ListPointPlot3D}[\mathbf{data},\mathbf{BoxRatios}\to 1,\mathbf{PlotStyle}\to\mathbf{PointSize}[0.02]]$$



Вот два примера множества точек, расположение которых указывает на поверхность (предыдущий рисунок в центре)

$$\mathbf{data = Flatten[Table[\{x, y, Sin\left[\frac{x^2}{4}+\frac{y^2}{9}\right] / \left(\frac{x^2}{4}+\frac{y^2}{9}\right)\},\ \{x, -8, 8, 0.25\}, \{y, -8, 8, 0.25\}], 1];//Quiet}$$

$$\mathbf{ListPointPlot3D[data]}$$

и предыдущий рисунок справа

$$\mathbf{ListPointPlot3D[Table[Cos[j + i], \{i, -\pi, \pi, 0.25\}, \{j, -\pi, \pi, 0.25\}]]}$$

Функция `ListSurfacePlot3D` принимает список координат точек в пространстве и пытается нарисовать поверхность, проходящую через эти точки

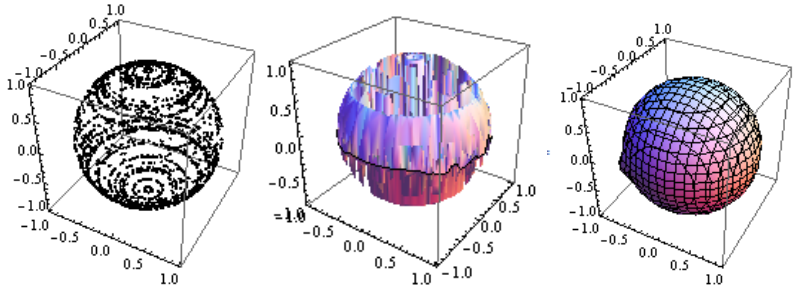
В следующем примере мы создаем множество точек, расположенных на сфере в случайном порядке, и строим поверхность, проходящую через это множество с помощью функций `ListPlot3D` и `ListSurfacePlot3D`.

$$\mathbf{dat = Flatten[Table[\{Cos[u]Cos[v], Cos[u]Sin[v], Sin[u]\},\ \{u, RandomReal[\{0, 2\pi\}, 50]\}, \{v, RandomReal[\{0, 2\pi\}, 50]\}], 1];}$$

$$\mathbf{ListPointPlot3D[dat, BoxRatios \to 1]}$$

$$\mathbf{ListPlot3D[dat, BoxRatios \to \{1, 1, 1\}, Mesh \to Full]}$$

$$\mathbf{ListSurfacePlot3D[dat, BoxRatios \to 1]}$$

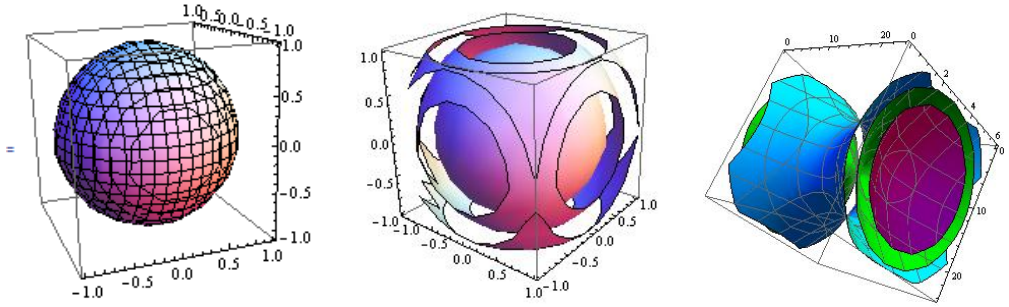


Здесь на левом рисунке показано сгенерированное множество точек, на среднем – поверхность, которая получается при попытке функции `ListPlot3D` представить это множество (она учитывает порядок точек в множестве/списке), на правом – поверхность, аппроксимирующая это множество (без учета порядка точек).

Функция `ListContourPlot3D` выполняет работу аналогичную функции `ListSurfacePlot3D`, но умеет строить несколько поверхностей. В формате `ListContourPlot3D[{{x1, y1, z1}, ...}, Contours->{c}]` она принимает массив координат точек – список вида $\{\{x_1, y_1, z_1\},...\}$, создает список $\{\{x_1, y_1, z_1, c\},...\}$ и путем интерполяции пытается представить эти значения в форме поверхности уровня некоторой функции $f(x, y, z) = c$. Если опция `Contours->{c1, c2, ..., cn}` содержит список значений, то функция

`ListContourPlot3D` пытается построить поверхности уровня для нескольких значений c_i. Например для предыдущего набора точек, случайно расположенных на поверхности сферы, получаем

ListContourPlot3D[dat, Contours → {0}] (* следующий рисунок слева *)



Использование опции `Contours` с несколькими значениями создает несколько поверхностей

ListContourPlot3D[dat, Contours → {0, 0.2, 0.4}, Mesh → None]

(предыдущий рисунок в центре). Предыдущий рисунок справа создан следующими командами

```
data = Table[x^2 + y^2 - z^2, {x, -3, 3}, {y, -3, 3, 0.25}, {z, -3, 3, 0.25}];
ListContourPlot3D[data, Contours → {-3, 0, 3},
    ContourStyle → {Purple, Green, Cyan},
    Mesh → 5, MeshStyle → GrayLevel[0.4]]
```

2.3.4 Анимация и манипуляторы

Анимация – это последовательность изображений, которые быстро сменяют друг друга, в результате чего появляется движение. Любую последовательность графиков можно анимировать. Анимация в *Mathematica* 5 и старших версиях немного отличается.

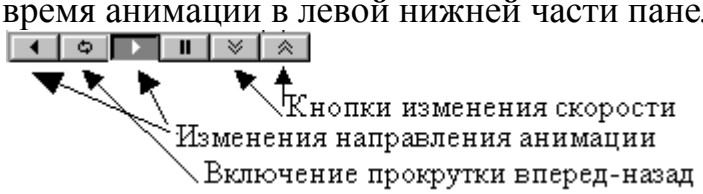
Опишем выполнение анимации в *Mathematica* 5. Для этого надо создать несколько подряд идущих графиков. Например, следующая команда будет генерировать последовательность из шести графиков, немного отличающихся друг от друга.

```
Do[Plot[Sin[n x],{x,0,2 Pi}],{n,1,2,0.2}]
```

Команда `Do` – универсальная итерационная команда/функция. В этом примере функция `Plot` повторяется шесть раз путем изменяя значение `n` от 1 до 2 с шагом 0.2.

Чтобы увидеть анимацию необходимо щелкнуть мышью на большой квадратной скобке, которая объединяет все графики. Группа графических ячеек (cells), содержащая изображения, будет помечена (скобка справа превратится в толстую черную линию). После этого необходимо выполнить команду меню запуска анимации Cell - Animate Selected Graphics или нажать комбинацию клавиш Ctrl-Y. Можно просто выполнить двойной щелчок мышью по любой картинке последовательности. Перемещения появляются в одной из ячеек, содержащей графическое изображение. Вы можете изменить скорость и направление движения анимации, нажимая кнопки, которые появляются во

время анимации в левой нижней части панели StatusBar.

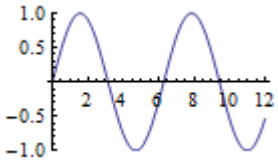


Вы можете создавать 3D анимацию таким же образом, используя функцию `Plot3D` вместо `Plot`.

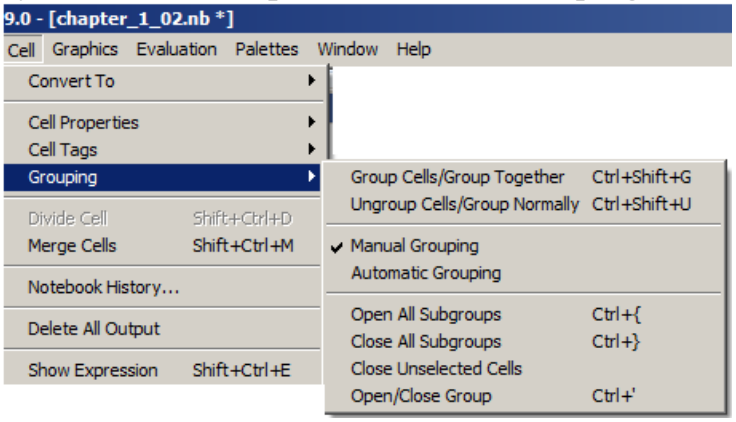
```
Do[Plot3D[ Sin[x y], {x,-n,n},{y,-n,n}, PlotRange->{-1,1},
                                    PlotPoints->20 ], {n,1,3,0.5}]
```

В новых версиях Mathematica $(version \geq 6)$ необходимо использовать функцию `Print`. Она должна обрамлять графическую функцию, строящую изображения.

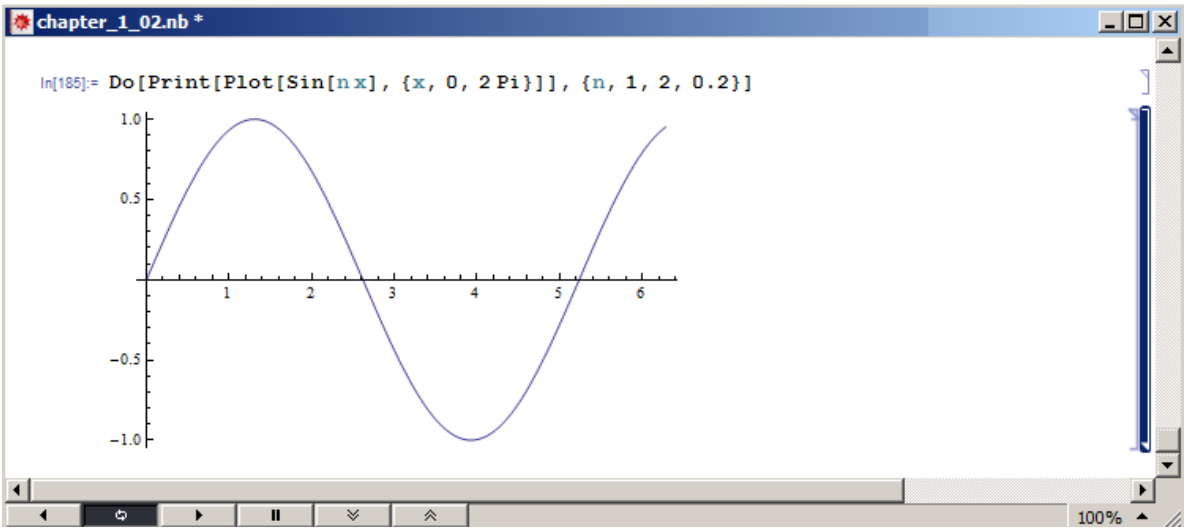
$$\mathbf{Do[Print[Plot[Sin[}\boldsymbol{ax}\mathbf{]}, \{\boldsymbol{x}, \mathbf{0}, \mathbf{12}\}, \mathbf{PlotRange} \rightarrow \{-\mathbf{1}, \mathbf{1}\}]], \{\boldsymbol{a}, \mathbf{1}, \mathbf{3}, \mathbf{0.1}\}]$$



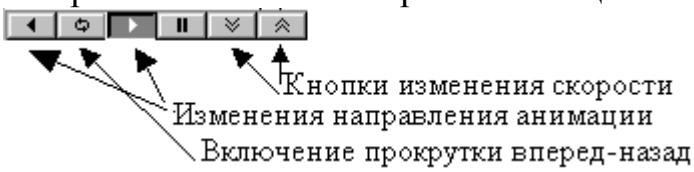
Лучший результат анимации получается, когда есть охватывающая все графики скобка. Если ее нет, то можно вручную перегруппировать секции. Для этого можно использовать меню Cell – Grouping – Group Cells и Cell – Grouping – Ungroup Cells при установленном режиме Cell – Grouping – Manual Grouping.



После создания скобки, охватывающей все секции с графиками, выполните по ней двойной щелчок; это свернет последовательность секций, оставив видимой только первую секцию. Теперь выделите скобку этой секции (или охватывающую все графики скобку) и и из меню Graphics - Rendering - Animate Selected Graphics запускайте анимацию (или нажмите комбинацию клавиш Ctrl-Y). Внизу появляется панель управления анимацией. Вид окна документа в момент анимации показан на следующем рисунке.



Вы можете изменить скорость и направление движения, нажимая кнопки, которые появляются во время анимации в левой нижней части окна документа.



Вы можете создавать 3D анимацию таким же образом, используя команду `Plot3D` вместо `Plot`.

```
Do[Print[
    Plot3D[Sin[xy], {x, -n, n}, {y, -n, n}, PlotRange → {-1, 1},
      PlotPoints → 20]],
{n, 1, 3, 0.5}]
```

Не забывайте перегруппировывать секции, иначе анимация будет захватывать ненужные секции!

Вы можете создавать анимацию с любой графической функцией *Mathematica*. Например,

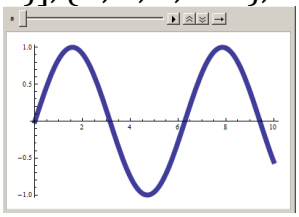
```
Do[Print[
    DensityPlot[Sin[xy], {x, -n, n}, {y, -n, n},
      PlotRange → {-1, 1}, PlotPoints → 20]],
{n, 1, 3, 0.5}]
```

В *Mathematica* 6 и последующих версиях для выполнения анимации предназначена функция `Animate`. Она имеет синтаксис

`Animate[expr, {t, t1, t2}],`

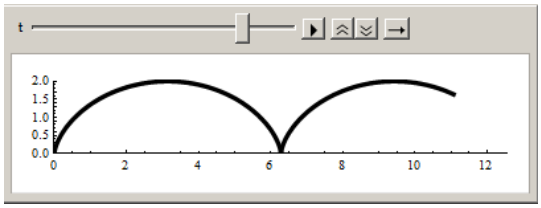
где кадр от кадра отличается значением параметра *t*, который изменяется в диапазоне от t_1 до t_2. Функция создает специальную панель, в которой выполняются все построения

```
Animate[Plot[Sin[xa], {x, 0, 10}], {a, 1, 5, 0.1}, AnimationRunning → False]
```



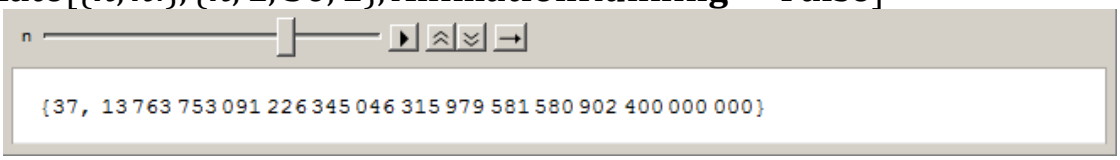
```
x[t_] = t - Sin[t];
```

```
y[t_] = 1 - Cos[t];
Animate[
    ParametricPlot[{x[τ], y[τ]}, {τ, 0, t}, PlotRange → {{0, 4π}, {0, 2}},
                                    PlotStyle → {Thickness[0.01], Black}],
    {t, 0.1, 4π, 0.1}]
```



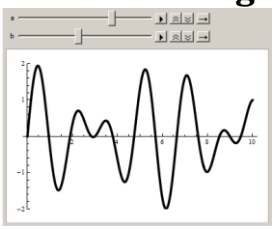
Опция `AnimationRunning→False` используется для того, чтобы сразу после создания панели, анимация не начинала работать. Выражение `expr` может быть не графическим объектом.

```
Animate[{n, n!}, {n, 1, 50, 1}, AnimationRunning → False]
```



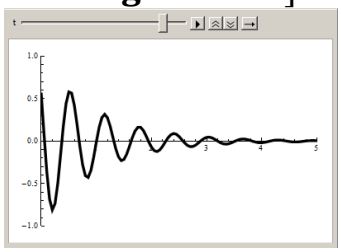
Разрешается использовать несколько параметров анимации

```
Animate[
        Plot[Sin[ax] + Sin[bx], {x, 0, 10}, PlotRange → 2,
                        PlotStyle → {Thickness[0.01], Black}],
        {a, 1, 5}, {b, 1, 5}, AnimationRunning → False]
```

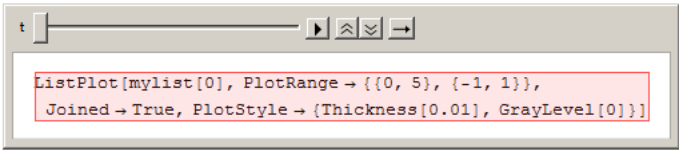


При построении анимации можно использовать любые графические функции.

```
mylist[t_] = Table[{x, Cos[10x - t]Exp[-x]}, {x, 0, 5, 0.05}];
Animate[
    ListPlot[mylist[t], PlotRange → {{0, 5}, {-1, 1}}, Joined → True],
    {t, 0, 6, 0.1}, AnimationRunning → False]
```



Важной опцией у функции `Animate` является опция `Initialization`. После выполнения предыдущего примера сохраните документ в файл, закройте окно документа и Математику, а потом откройте систему и файл. Вполне вероятно, что после открытия файла панель анимации будет пуста или подкрашена в розовый цвет, являющийся признаком ошибки. Например, она может иметь вид



Если вы выполните код секции, то все станет нормально, но до этого такая панель анимации будет вас раздражать. Дело в том, что панель анимации работает в оболочке системы (Front End), а не в ее ядре. Когда система загружает файл, ядро автоматически не вычисляет все выражения и функции, созданные и находящиеся в файле. Поэтому функция `Animate` после повторной загрузки файла не распознает объект `mylist[t]` (код выражения, создающего эту функцию еще не находится в рабочем пространстве системы) и оболочка системы не может правильно заполнить панель анимации данными. Если вы выполните код секции, то в памяти системы появится объект/функция `mylist[t]` и графическое поле панели анимации будет обновлено. Опция `Initialization` предназначена для автоматического выполнения кода, указанного в этой опции, при первом появлении панели анимации в окне документа. Она имеет синтаксис `Initialization:→(код)`, где `код` – это любая последовательность команд системы. Таким образом, для предыдущего примера правильным будет следующий код

```
Animate[ListPlot[mylis[t],
                 PlotRange → {{0, 5}, {-1, 1}}, Joined → True,
                 PlotStyle → {Thickness[0.01], Black}],
{t, 0, 6, 0.1}, AnimationRunning → False,
Initialization :→ (mylis[t_]
          = Table[{x, Cos[10x - t]Exp[-x]}, {x, 0, 5, 0.05}])]
```

Здесь определение функции `mylist[t]` перенесено в тело опции `Initialization`, которое выполняется при появлении окна панели анимации в видимой области документа. Заметим, что опция `Initialization` имеется у многих функций, которые выполняются в оболочке системы.

Также корректно будет работать следующий код

```
DynamicModule[{mylst},
  mylst[t_] = Table[{x, Cos[10x - t]Exp[-x]}, {x, 0, 5, 0.05}];
  Animate[ListPlot[mylst[t],
                   PlotRange → {{0, 5}, {-1, 1}}, Joined → True,
                   PlotStyle → {Thickness[0.01], Black}],
  {t, 0, 6, 0.1}, AnimationRunning → False]]
```

Функция `DynamicModule` используется для того, чтобы ограничить область действия динамических переменных. О них мы будем говорить отдельно в других частях нашего пособия. Функция имеет следующий синтаксис

`DynamicModule[{x,y,...}, `*выражение*`]`

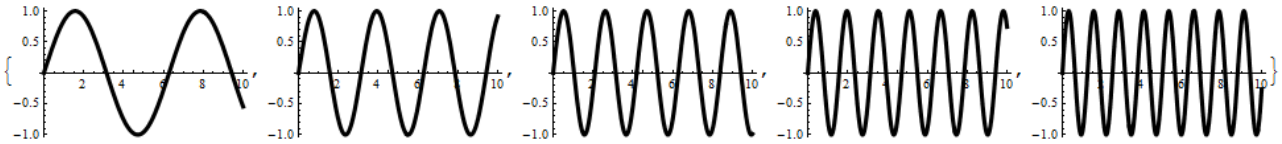
или

`DynamicModule[{x=x`$_0$`,y=y`$_0$`,...}, `*выражение*`]`

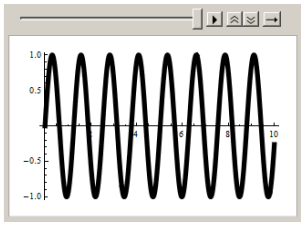
и ограничивает динамическую область для переменных *x*, *y*,.... Второй способ задает начальные значения динамических переменных. В нашем примере функция `DynamicModule` создает блок кода, в котором имя `mylst` является динамической защищенной в пределах этого блока переменной. Значения/выражения локальных переменных динамического блока сохраняются в файле и существуют в следующих сессиях работы с этим файлом.

Есть еще один способ создания анимации с помощью функции `ListAnimate`. Эта функция принимает список графических объектов и воспроизводит их последовательно в панели анимации. Это похоже на анимацию с использованием способа Do[Print[..., только последовательность графических объектов воспроизводится в панели анимации.

```
tp = Table[Plot[Sin[nx], {x, 0, 10}], {n, 5}]
```



```
ListAnimate[tp]
```



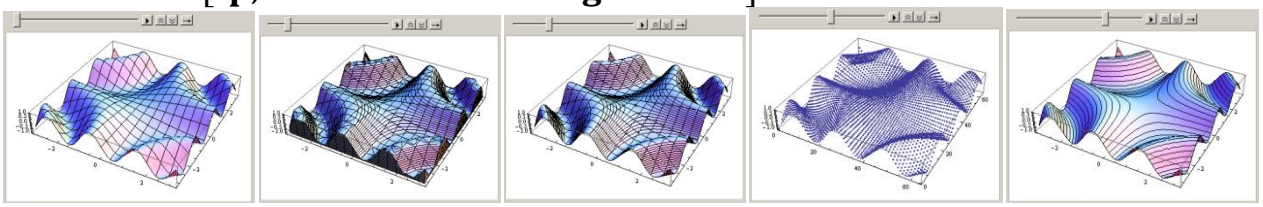
Обычное использование функции `ListAnimate` будет с точкой с запятой после вызова функции `Table`. Это необходимо, чтобы предотвратить отображение большого количества графиков. Кроме того, как правило, будет использоваться опция `AnimationRunning → False`. Таким образом, приведенные команды чаще будут использоваться следующим образом

```
tp = Table[Plot[Sin[nx], {x, 0, 10}], {n, 5}];
ListAnimate[tp, AnimationRunning → False]
```

В примере список **tp** состоял из графиков (графических объектов) и создавался с помощью функции `Table`. Но список из графических объектов можно создавать любым способом, например, перечислением объектов в списке. В следующем коде строится несколько различных графиков одной и той же функции $z = \sin(x\,y)$, которые затем последовательно воспроизводятся функцией `ListAnimate`.

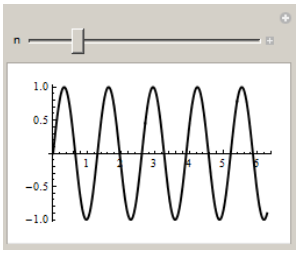
```
p1 = Plot3D[Sin[xy], {x, -π, π}, {y, -π, π}, BoxRatios → {2π, 2π, 1}];
p2 = RegionPlot3D[Sin[xy] > z, {x, -π, π}, {y, -π, π}, {z, -1, 1},
                  BoxRatios → {2π, 2π, 1}];
p3 = ContourPlot3D[Sin[xy] - z == 0, {x, -π, π}, {y, -π, π}, {z, -1, 1},
                  BoxRatios → {2π, 2π, 1}];
p4 = ListPointPlot3D[Table[Sin[i * j], {i, -π, π, 0.1}, {j, -π, π, 0.1}],
                  BoxRatios → {2π, 2π, 1}];
```

```
p5 = Plot3D[Sin[xy], {x, -π, π}, {y, -π, π}, BoxRatios → {2π, 2π, 1},
                MeshFunctions → {#1 * #2&}, Mesh → 50];
tp = {p1, p2, p3, p4, p5};
ListAnimate[tp, AnimationRunning → False]
```



Кроме панели анимации в *Mathematica version* ≥ 6 есть другие панели, динамически управляющие своим содержимым, например, графиком

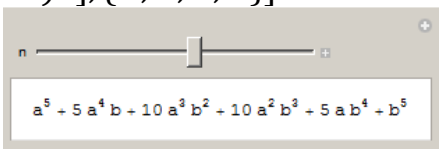
```
Manipulate[
    Plot[Sin[nx], {x, 0, 2Pi}, PlotStyle → {Thickness[0.01], Black}],
    {n, 1, 20}]
```



Передвигая бегунок панели `Manipulate`, вы будете менять значение параметра `n`, к которому он «привязан».

Содержимое панели `Manipulate` не обязано быть графическим объектом

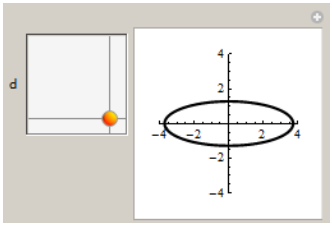
```
Manipulate[Expand[(a + b)^n], {n, 1, 8, 1}]
```



Можно создать 2D манипулятор. Перемещение его указателя в пределах прямоугольного окна управляет значением (парой чисел), связанным с манипулятором.

```
Manipulate[
  ParametricPlot[{d[[1]]Cos[t], d[[2]]Sin[t]}, {t, 0, 2Pi},
          PlotRange → {{-4, 4}, {-4, 4}},
          PlotStyle → {Thickness[0.01], Black}],
  {d, {1, 1}, {4, 4}}, ControlPlacement → Left]
```

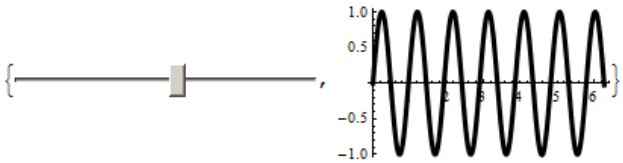
Здесь переменная `d` представляет список из двух чисел. Эта пара чисел представляет координаты указателя манипулятора и определяется его положением. Список вида `{d,{1,1},{4,4}}` задает минимальные значения координат указателя по обеим направлениям {1, 1} и максимальные значения {4,4}.



Функция `Manipulate` очень богатая на различные представления. Только описание ее возможностей могло бы занять целую главу.

Кроме `Manipulate` существуют другие элементы, которые могут динамически изменять параметры. Например

```
DynamicModule[{a =. 5},
    {Slider[Dynamic[a]], Dynamic[Plot[Sin[12a t], {t, 0, 2π}]]}]
```



Здесь функция `Dynamic` создает динамическую переменную `a`, привязанную к бегунку (Slider). Его перемещение интерактивно меняет значение переменной `a`. Вторая функция `Dynamic[Plot[...]]` следит за изменениями динамических переменных (в нашем случае за значением переменной `a`), и обновляет свое содержимое в соответствии с новым значением `a`. Заключение кода внутрь функции `DynamicModule` разрывает связь динамической переменной `a`, например, с переменной `a` предыдущего примера, но оставляет связь с переменной `a` функции `Plot`, которая расположена в блоке/теле функции `DynamicModule`.

Функция `LocatorPane` предоставляет область с «локатором», который можно перемещать с помощью мыши. В формате

```
LocatorPane[Dynamic[pt], тело]
```

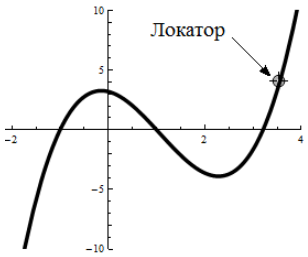
она позволяет динамически менять положение указателя и использовать его координаты в теле (последовательности команд).

В следующем примере мы строим график кубического полинома с двумя корнями в точках ± 1, который проходит через точку (a,b). Уравнение такого полинома имеет вид

$$f(x)=\left(x^2-1\right)\cdot\left(x-\frac{a^3-a-b}{a^2-1}\right)$$

Точка (a,b) привязана к «локатору». Его перемещение меняет координаты точки (a,b) и, тем самым, меняет график кубического полинома.

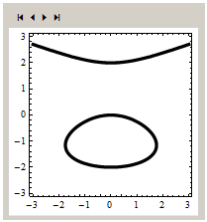
```
DynamicModule[{a = 0, b = 0, f},
    f[x_, a_, b_] = (x^2 - 1) (x - (a^3-a-b)/(a^2-1));
    LocatorPane[Dynamic[{a, b}],
                Dynamic[Plot[f[x, a, b], {x, -4, 4},
                             PlotRange → {{-4, 4}, {-10, 10}},
                             PlotStyle → {Black, Thickness[0.01]}]]]]
```



Захватите мышью «локатор» и перемещайте его. Кривая будет автоматически перерисовываться в соответствии с новым положением точки (a,b). Назначение функций `Dynamic` и `DynamicModule` такое же, как и в предыдущем примере.

Функция `SlideView` отображает панель, в которой можно дискретно менять значения какой – либо переменной. И если она (переменная) входит в выражение по которому строится график внутри этой панели, то каждое ее изменение будет приводить к изменению графика.

```
SlideView[Table[
        ContourPlot[x^2 - y^3 == p y, {x, -3, 3}, {y, -3, 3}],
        {p, -4, 4}]]
```



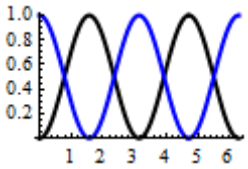
Щелкая по управляющим кнопкам «слайдера», вы будете менять значение параметра `p` в диапазоне от –4 до 4 с шагом 1, а следовательно и выражение, график которого отображается в панели.

Имеется еще большое количество управляющих элементов, которые позволяют динамически менять графику, расположенную внутри их области ответственности. Например, еще имеются функции `FlipView`, `MenuView`, `PopupMenu`. С подробным описанием этих и других подобных функций вы можете познакомиться по справочной системе.

2.3.5 Комбинирование графиков

Все графические функции возвращают/создают графические объекты. Уже созданные, такие объекты можно отображать в общем координатном пространстве. Так функция `Show` позволяет отобразить несколько графиков одновременно или показать тот же график, но с другими опциями. Фактически она накладывает один графический объект на другой.

```
ps = Plot[Sin[x]^2, {x, 0, 2Pi}, DisplayFunction → Identity];
pc = Plot[Cos[x]^2, {x, 0, 2Pi}, DisplayFunction → Identity];
Show[ps, pc, DisplayFunction → $DisplayFunction]
```

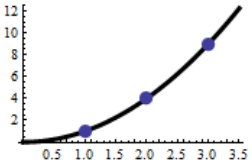


Установка опции `DisplayFunction→Identity` отменяет вывод графика на экран, хотя он создается в памяти. Затем функция `Show` отображает созданные графические объекты (здесь `ps` и `pc`) в окне документа. Обратно, опция `DisplayFunction→$DisplayFunction` при вызове функции `Show` включает отображение графических объектов. Этот способ работает во всех версиях *Mathematica*, начиная с версии 4. В *Mathematica version* ≥ 6 можно не использовать опцию `DisplayFunction→ Identity`. Достаточно завершить вызов графической функции точкой с запятой. Поэтому создание и рисование двух последних графических объектов можно выполнить короче

```
ps = Plot[Sin[x]^2, {x, 0, 2Pi}];
pc = Plot[Cos[x]^2, {x, 0, 2Pi}];
Show[ps, pc]
```

Графики могут быть созданы различными графическими функциями

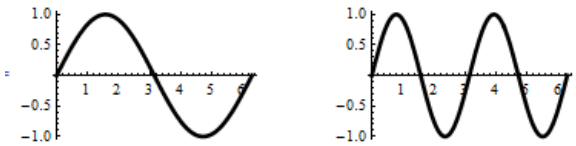
```
Show[Plot[x^2, {x, 0, 3.5}, PlotStyle → {Thickness[0.01], Black}],
     ListPlot[{1, 4, 9}, PlotStyle → PointSize[0.04]]]
```



При использовании функции `Show` необходимо побеспокоиться о выравнивании масштабов графиков, налагаемых друг на друга.

В *Mathematica* 5 функция `GraphicsArray` создает графический объект, составленный из нескольких других, расположенных рядом (в одну строку).

```
p1 = Plot[Sin[x], {x, 0, 2Pi}, DisplayFunction → Identity];
p2 = Plot[Sin[2x], {x, 0, 2Pi}, DisplayFunction → Identity];
Show[GraphicsArray[{p1, p2}]]
```



Изменить расстояние между графиками в окне документа можно с помощью опции `GraphicsSpacing`.

```
Show[GraphicsArray[{p1, p2}, GraphicsSpacing → 0.5]]
```

В *Mathematica version* ≥ 6 появились новые функции. Для отображения нескольких графиков в ряд используется функция `GraphicsRow`. Ей в качестве аргумента передается список имен графических объектов. Например, чтобы отобразить два графических объекта `ps` и `pc` из предыдущего примера в одной строке можно выполнить команду

```
Show[GraphicsRow[{ps, pc}]]
```

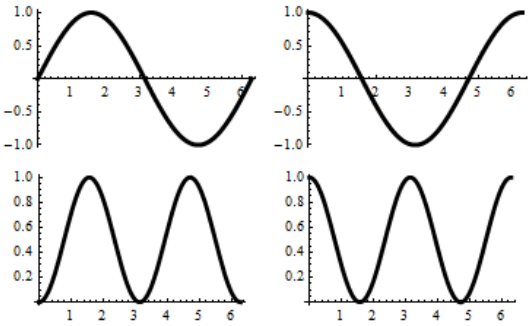
При этом использовать функцию Show необязательно

GraphicsRow[{ps, pc}]

Имеется аналогичная функция GraphicsColumn, располагающая графики один под одним.

Функция GraphicsGrid отображает графики таблично – в несколько строк и столбцов. Ее аргументом должен быть список списков графических объектов.

ps1 = Plot[Sin[x], {x, 0, 2Pi}];
pc1 = Plot[Cos[x], {x, 0, 2Pi}];
ps2 = Plot[Sin[x]^2, {x, 0, 2Pi}];
pc2 = Plot[Cos[x]^2, {x, 0, 2Pi}];
GraphicsGrid[{{ps1, pc1}, {ps2, pc2}}]



Есть еще функция Grid, которая в виде таблицы может отображать не только графические объекты. Применительно к графикам ps1, pc1, ps2, pc2 из предыдущего примера, она может быть использована следующим образом

Grid[{{ps1, pc1}, {ps2, pc2}}, Frame → All]

В результате будет построены те же графики, что и выше, но оформленные в виде таблицы (с рамкой и разделительными линиями).

Функция Grid обладает большими возможностями по оформлению объектов различного типа в виде таблицы. Например

Grid[{{(a + b)^2, (a + b)^3}, {Expand[(a + b)^2], Expand[(a + b)^3]}}, Frame → All]

$(a+b)^2$	$(a+b)^3$
$a^2+2ab+b^2$	$a^3+3a^2b+3ab^2+b^3$

Подробнее с ее возможностями вы можете познакомиться по справочной системе.

2.3.6 Графические примитивы.

Примитивами называют стандартные геометрические фигуры. Они создаются с помощью функций Graphics[primitives,options] и Graphics3D[primitives,options]. Одним из аргументов этих функций являются имена примитивов, а точнее функции создания примитивов. Они создают математические объекты, которые функциями Graphics и Graphics3D преобразуются в графические объекты. Эти функции добавляют в структуру объекта примитива стили (цвет, толщину линии и т.д.). Полученные графические объекты могут быть отображены в окне документа с

помощью функции Show или другим способом. Основными двумерными примитивами являются:

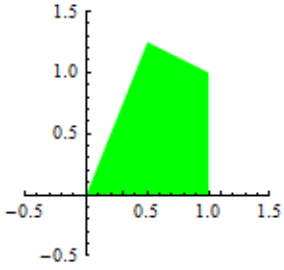
`Circle[{x,y},r]`	окружность с центром в точке `{x,y}` и радиусом `r`;
`Disk[{x,y},r]`	круг (область) с центром в точке `{x,y}`, радиусом `r`;
`Line[{{x1,y1},…}]`	ломаная с вершинами в точках $\{\{x_1, y_1\}, \ldots\}$;
`Point[{x,y}]`	точка с координатами `{x,y}`;
`Polygon[{{x1,y1},…}]`	замкнутый многоугольник (область) с вершинами в точках $\{\{x_1, y_1\}, \ldots\}$;
`Rectangle[{x1,y1},{x2,y2}]`	прямоугольник (область), определяемый двумя диагонально расположенными вершинами;
`Text["string",{x,y}]`	текстовая строка, расположенная в точке с координатами `{x,y}`;

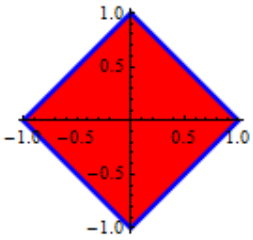
Приведем примеры использования этих примитивов.

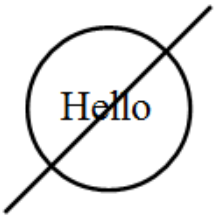
```
Show[Graphics[Disk[{0, 0}, 1]]]
```

Чтобы добавить цвет функции Graphics надо передать первым аргументом директиву задания цвета RGBColor. Директивы и примитив, к которому они относятся, следует объединять в список. При этом директива должна стоять перед примитивом.

```
Show[Graphics[{RGBColor[0, 1, 0],
                Polygon[{{0, 0}, {1, 0}, {1, 1}, {0.5, 1.25}}]}],
  PlotRange → {{-0.5, 1.5}, {-0.5, 1.5}}, Axes → True,
  AspectRatio → Automatic]   (* следующий рисунок слева *)
```







В *Mathematica version* ≥ 6 при отображении графического примитива можно не использовать функцию Show – достаточно функции Graphics (или Graphics3D). Например две следующие строки кода эквивалентны

```
Show[Graphics[p]]
Graphics[p]
```

Вот пример построения красного квадрата с синим контуром без использования функции Show.

```
vertices = {{0, -1}, {1, 0}, {0, 1}, {-1, 0}, {0, -1}};
p = Graphics[{EdgeForm[{Blue, Thickness[0.02]}],
              RGBColor[1, 0, 0], Polygon[vertices]}]
```

Графическая директива `EdgeForm[{Blue,Thickness[0.02]}]` определяет, что контур последующего примитива будет иметь заданный цвет и толщину.

Если на рисунок надо наложить какие либо дополнительные графические элементы, например, показать оси координат, то все же следует использовать функцию `Show`.

```
Show[p, AspectRatio → Automatic, Axes → True] (* пред. рис. в центре *)
```

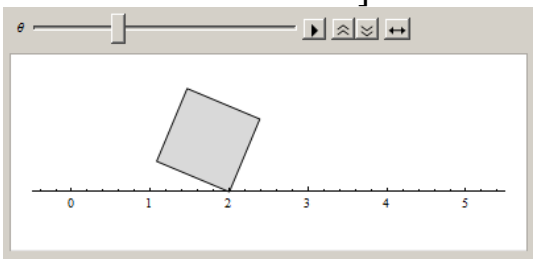
Вот пример использования текстового примитива совместно с другими примитивами.

```
g1 = Graphics[{Thickness[.02], Line[{{-1, -1}, {1, 1}}]}];
g2 = Graphics[{Thickness[.02], Circle[{0, 0}, 0.8]}];
g3 = Graphics[Text[Style["Hello", Large]], {0, 0}];
Show[g1, g2, g3] (* предыдущий рисунок справа *)
```

Реже используемыми двумерными примитивами и связанными с ними функциями являются `Arrow`, `BezierCurve`, `BSplineCurve`, `FilledCurve`, `GraphicsComplex`, `GraphicsGroup`, `JoinedCurve`, `Locator`, `Raster`.

Графические примитивы можно использовать для создания анимации. В следующем примере мы создаем анимацию катящегося по прямой квадрата

```
Animate[With[{q = Quotient[θ, π/2], m = Mod[θ, π/2]},
        Graphics[{EdgeForm[Black], LightGray,
                  Rotate[Rectangle[{q, 0}], -m, {q + 1, 0}]},
                 Axes → {True, False},
                 PlotRange → {{-.5, 5.5}, {-0.5, 1.5}}]],
{θ, 0, 2Pi}, AnimationRunning → False,
AnimationDirection → ForwardBackward]
```



Поясним некоторые моменты приведенного кода. Функция `With[{q=expr1,m=expr2}, expr(q,m)]` в выражении `expr(q,m)` выполняет замену идентификаторов `q` и `m` значениями `expr1` и `expr2`. Функции `Quotient`[θ,π/2] вычисляет целую часть от деления первого аргумента θ на второй аргумент π/2, а `Mod`[θ,π/2] вычисляет остаток деления θ на π/2. Графическая директива `EdgeForm[Black]` определяет, что контур последующего примитива (закрашенного в светло серый цвет квадрата) должен иметь заданный цвет. Функция `Rectangle[{q,0}]` создает примитив – квадрат с левой нижней вершиной в точке {q,0} и стороной 1 (по умолчанию). Функция `Rotate[Rectangle[{q,0}],-m,{q+1,0}]` поворачивает графический примитив `Rectangle[{q,0}]` на угол –`m` вокруг точки с координатами `{q+1,0}` , т.е вокруг правой нижней вершины квадрата.

Положительное направление угла отсчитывается против часовой стрелки, поэтому угол поворота равен –m. Опция `AnimationDirection` определяет направление анимации.

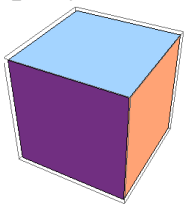
Заметим, что функция Rotate может поворачивать не только графические объекты. Например в следующем коде мы поворачиваем текст.

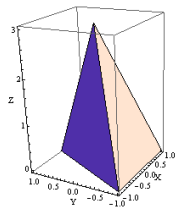
Rotate["**Hello**", $\boldsymbol{\pi/4}$]

Функция `Graphics3D` преобразует трехмерные примитивы в трехмерные графические объекты, добавляя в структуру примитива стили (цвет, толщину и т.д.). Основными трехмерными примитивами являются:

`Cuboid[{x1,y1,z1},…]`	параллелепипед (3D область), определяемый двумя диагонально расположенными вершинами;
`Line[{{x1,y1,z1},…}]`	ломаная в пространстве с вершинами в точках $\{\{x_1, y_1, z_1\}, \ldots\}$;
`Point[{x,y,z}]`	точка с координатами `{x,y,z}`;
`Polygon[{{x1,y1,z1},…}]`	многогранная двумерная область в пространстве с вершинами в точках $\{\{x_1, y_1, z_1\}, \ldots\}$;
`Text[“string”, {x, y, z}]`	текстовая строка, расположенная в точке с координатами `{x,y,z}`;

g1 = Graphics3D[Cuboid[{1, 1, 1}]];
Show[g1] (* следующий рисунок слева *)





Можно рисовать сложные фигуры, комбинируя уже имеющиеся примитивы. В следующем примере мы создаем пирамиду с вершинами в точках `(1,-1,0)`, `(0,1,0)`,`(-1,-1,0)`,`(0,0,3)`.

```
piramid = {Polygon[{{1, -1, 0}, {0, 1, 0}, {-1, -1, 0}}],
           Polygon[{{1, -1, 0}, {0, 0, 3}, {0, 1, 0}}],
           Polygon[{{0, 1, 0}, {0, 0, 3}, {-1, -1, 0}}],
           Polygon[{{-1, -1, 0}, {0, 0, 3}, {1, -1, 0}}]};
Show[Graphics3D[piramid], Axes → Automatic, AxesLabel → {"X", "Y", "Z"}]
```

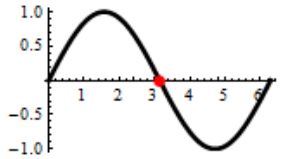
(предыдущий рисунок справа). Тот же графический объект может быть получен с использованием одной функции `Polygon`

```
piramid = Polygon[{{{1, -1, 0}, {0, 1, 0}, {-1, -1, 0}},
                   {{1, -1, 0}, {0, 0, 3}, {0, 1, 0}},
                   {{0, 1, 0}, {0, 0, 3}, {-1, -1, 0}},
                   {{-1, -1, 0}, {0, 0, 3}, {1, -1, 0}}}];
```

Show[**Graphics3D**[**piramid**], **Axes** → **Automatic**, **AxesLabel** → {"**X**", "**Y**", "**Z**"}]
В случае, когда параметрические уравнения поверхностей неизвестны, примитивы являются самым надежным способом создания графического объекта, представляющего поверхность. Отметим однако, что параметрические уравнения поверхности куба или пирамиды можно построить и мы приводили их ранее.

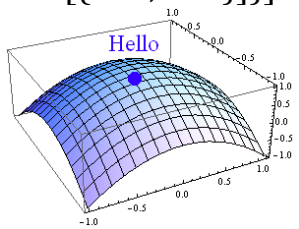
Графические примитивы можно отображать не только с помощью функции `Show`. Опции `Epilog` и `Prolog` позволяют отображать примитивы на графиках функций. Опция `Prolog` позволяет отобразить примитив перед рисованием графика, а `Epilog` – после.

Plot[**Sin**[x], {x, **0**, **2Pi**},
Epilog → {**PointSize**[**0.05**], **Hue**[**1**], **Point**[{**Pi**, **Sin**[**Pi**]}]},
PlotStyle−> {**Thickness**[**0.02**], **Black**}]



На трехмерный график можно нанести двумерный примитив. При этом его положение следует задавать в относительных координатах (0,1).

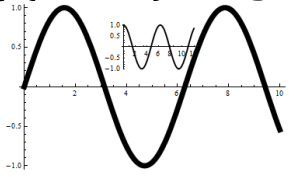
Plot3D[**1** − x^2 − y^2, {x, −**1**, **1**}, {y, −**1**, **1**},
Epilog → {**PointSize**[**0.05**], **Hue**[**0.7**], **Text**[**Style**["**Hello**", **Large**],
{**0.5**, **0.75**}], **Point**[{**0.5**, **0.6**}]}]



Здесь опция Epilog отображает два примитива: текстовый и точку. Список {0.5,0.75} определяет положение левой нижней точки текстового примитива, а список {0.5,0.6} определяет положение точки.

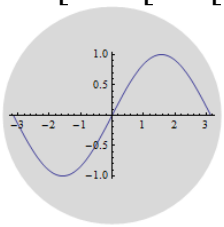
Полезной является функция `Inset`, создающая графический объект, который может быть вставлен в другой графический объект. Ее аргументами являются графический объект, например, график функции, и необязательные опции, управляющие размером вставляемого объекта и местом его расположения в родительском графическом объекте. Функция `Inset` может использоваться везде, где можно использовать графический примитив. Например, в следующем примере мы вставляем один график в другой

Plot[**Sin**[x], {x, **0**, **10**},
Epilog → **Inset**[**Plot**[**Cos**[x], {x, **0**, **12**}, **ImageSize** → **120**], {**5**, **0.5**}]]



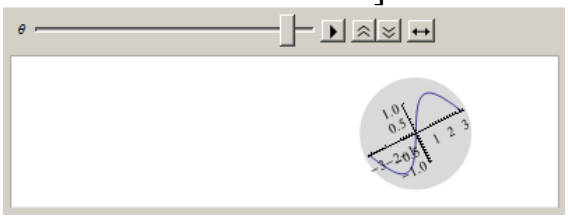
Следующий код рисует графический примитив `Disk[]` с вставленным в него графиком.

```
Graphics[{LightGray, Disk[ ], Inset[Plot[Sin[x], {x, -π, π}]]}]
```



В следующем примере мы создаем графический объект – диск, в который с помощью функции `Inset` вставлен график функции синус. Этот диск с графиком катится по прямой

```
Animate[Graphics[
    Rotate[{LightGray, Disk[{θ, 1}, 1],
    Inset[Plot[Sin[x], {x, -π, π}], {θ, 1}, {0, 0}, {2, Automatic}]}, -θ, {θ, 1}],
    PlotRange → {{-1, 8}, {0, 2}}],
{θ, 0, 2Pi}, AnimationRunning → False,
AnimationDirection → ForwardBackward]
```



Здесь следует пояснить дополнительные аргументы функции `Inset[Plot[Sin[x],{x,-π,π}],{θ,1},{0,0},{2,Automatic}]`. Второй ее аргумент `{θ,1}` определяет точку родительского графического объекта, в которую вставляется точка `{0,0}` дочернего объекта (графика). Последний аргумент `{2,Automatic}` определяет размер дочерней вставки. Задание размера в виде `{w, Automatic}` определяет, что вставляемый объект должен иметь ширину `w` в координатной системы родительского графического объекта.

2.4 Графические иллюстрации к решению прикладных задач

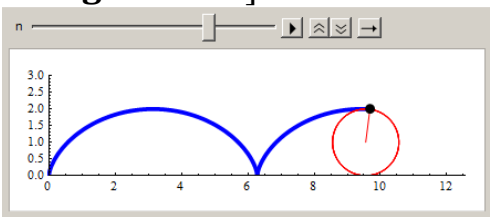
В этом разделе собраны примеры, демонстрирующие возможности графических функций системы *Mathematica* для представления решений прикладных задач. При этом мы не даем подробных постановок задач и, как правило, приводим только решение, графическое представление которого нас интересует. Если «физическая» постановка задачи вам неясна, то вы можете пропустить пример.

2.4.1. Несколько геометрических примеров

Пример 1. *Циклоида*. Траектория фиксированной точки окружности, которая катится по прямой, называется циклоидой. Если радиус окружности равен 1 и скорость движения ее центра $v=1$, то параметрическое уравнение циклоиды

будет иметь вид $x(t)=t-\sin t$, $y(t)=1-\cos t$. Следующий код строит анимацию качения окружности, движение фиксированной точки на окружности, и заметаемую точкой траекторию (циклоиду).

```
Animate[
  p1 = ParametricPlot[{t - Sin[t], 1 - Cos[t]}, {t, 0, n}
     PlotRange → {{0, 4π}, {0, 3}}, PlotStyle → {Blue, Thickness[0.01]}];
  p2 = Graphics[{Red, Line[{{n - Sin[n], 1 - Cos[n]}, {n, 1}}]}];
  p3 = ParametricPlot[{n - Sin[n * t], 1 - Cos[n * t]}, {t, 0, 2π},
                                                  PlotStyle → {Red}];
  p4 = Graphics[Disk[{n - Sin[n], 1 - Cos[n]}, 0.15]];
  Show[p1, p2, p3, p4],
{n, 1, 4π}, AnimationRunning → False]
```



Пример 2. Лента Мебиуса – простейшая неориентируемая поверхность. Она получается движением и вращением отрезка прямой вдоль замкнутой пространственной кривой. Пусть эта кривая будет окружностью радиуса `R`, а `n` обозначает количество полуоборотов отрезка при обходе кривой. Тогда уравнение поверхности можно записать в параметрическом виде

$$x(u,v)=\left(R+v\cos\left(\frac{nu}{2}\right)\right)\cos u$$

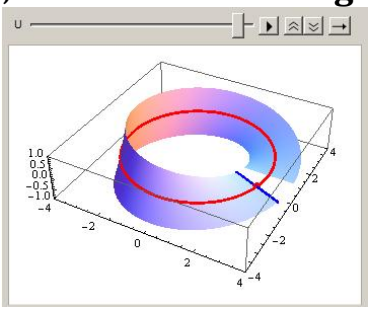
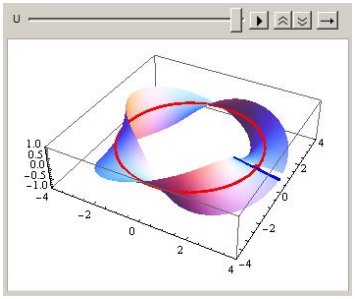
$$y(u,v)=\left(R+v\cos\left(\frac{nu}{2}\right)\right)\sin u$$

$$z(u,v)=v\sin\left(\frac{nu}{2}\right)$$

Вдоль окружности движется центр отрезка. Следующий код строит анимацию его движения и заметаемую им поверхность (лента Мебиуса).

```
DynamicModule[{R = 3, H = 2, n = 1, x, y, z, po, pt, pc, ps},
  x[u_, v_] = (R + vCos[nu/2])Cos[u];
  y[u_, v_] = (R + vCos[nu/2])Sin[u];
  z[u_, v_] = vSin[nu/2];
  po = ParametricPlot3D[{x[u, 0], y[u, 0], z[u, 0]}, {u, 0, 2π},
        PlotStyle → {Red, Thickness[0.01]},
        PlotRange → {{-4, 4}, {-4, 4}, {-1, 1}}];
  Animate[
    pt = Graphics3D[
        {Red, PointSize[0.03], Point[{x[U, 0], y[U, 0], z[U, 0]}]}];
```

```
pc = ParametricPlot3D[{x[U, v], y[U, v], z[U, v]}, {v, -H/2, H/2},
      PlotStyle → {Blue, Thickness[0.01]},
      PlotRange → {{-4, 4}, {-4, 4}, {-1, 1}}];
ps = ParametricPlot3D[{x[u, v], y[u, v], z[u, v]}, {u, 0, U}, {v, -H/2, H/2},
      AspectRatio → Automatic, Mesh → None, PlotPoints → {50, 15},
      PlotRange → {{-4, 4}, {-4, 4}, {-1, 1}}];
Show[pc, ps, po, pt],
{U, 0.01, 2π}, AnimationRunning → False]]
```

Здесь `H` обозначает ширину ленты (длину отрезка). При нечетном `n` получается неориентированная поверхность. На предыдущем рисунке слева показана лента Мебиуса, построенная при `n=1`, а справа – при `n=3`.

Заметим, что картинка ленты Мебиуса имеется в наборе примеров геометрических поверхностей, которую можно построить командой

```
ExampleData[{"Geometry3D", "MoebiusStrip"}]
```

□

Пример 3. Бутылка Клейна из лоскутов поверхностей.

Составим поверхность бутылки Клейна из 4-х лоскутов/поверхностей, параметрические уравнения которых достаточно просты.

Строим первый лоскут (следующий рисунок *a*)

```
a = 2.5; b = 1.5;
x1[u_, v_] = (a + bCos[u])Cos[v];
y1[u_, v_] = (a + bCos[u])Sin[v];
z1[u_, v_] = -aSin[u];
p1 = ParametricPlot3D[{x1[u, v], y1[u, v], z1[u, v]}, {u, 0, π}, {v, 0, 2π},
PlotStyle-> Opacity[0.5], Mesh → None]
```

Строим второй лоскут (следующий рисунок *b*)

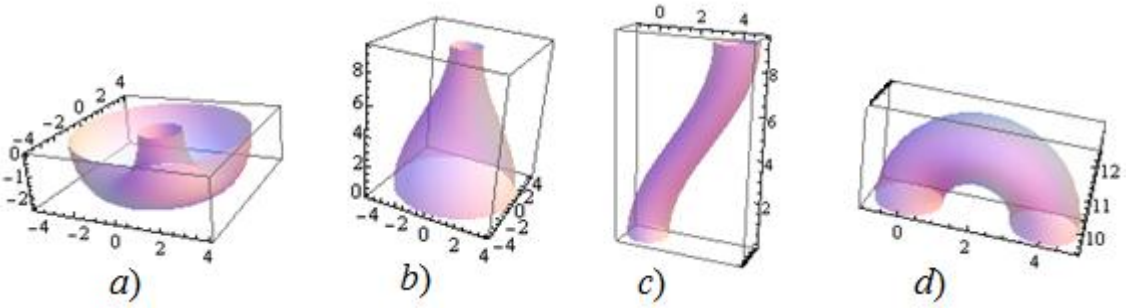
```
x2[u_, v_] = (a + bCos[u])Cos[v];
y2[u_, v_] = (a + bCos[u])Sin[v];
z2[u_, v_] = 3u;
p2 = ParametricPlot3D[{x2[u, v], y2[u, v], z2[u, v]}, {u, 0, π}, {v, 0, 2π},
PlotStyle-> Opacity[0.5], Mesh → None]
```

Строим третий лоскут (следующий рисунок *c*)

```
x3[u_, v_] = 2 - 2Cos[u] + Sin[v];
y3[u_, v_] = Cos[v];
z3[u_, v_] = 3u;
p3 = ParametricPlot3D[{x3[u, v], y3[u, v], z3[u, v]}, {u, 0, π}, {v, 0, 2π},
PlotStyle-> Opacity[0.5], Mesh → None]
```

Строим четвертый лоскут (следующий рисунок *d*)

$$\mathbf{x4[u\_, v\_] = 2 + (2 + Cos[v])Cos[u];}$$
$$\mathbf{y4[u\_, v\_] = Sin[v];}$$
$$\mathbf{z4[u\_, v\_] = 3\pi + (2 + Cos[v])Sin[u];}$$
$$\mathbf{p4 = ParametricPlot3D[\{x4[u, v], y4[u, v], z4[u, v]\}, \{u, 0, \pi\}, \{v, 0, 2\pi\},}$$
$$\mathbf{PlotStyle -> Opacity[0.5], Mesh \to None]}$$

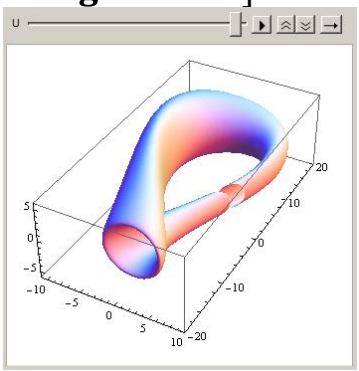


Теперь рисуем все сегменты/куски поверхности вместе.

$$\mathbf{Show[p1, p2, p3, p4, PlotRange \to All, Boxed \to False, Axes \to None]}$$

Пример 4. Бутылка Клейна. Известны примеры параметрических уравнений поверхностей, которые представляют бутылку Клейна. Здесь мы используем одно из таких уравнений.

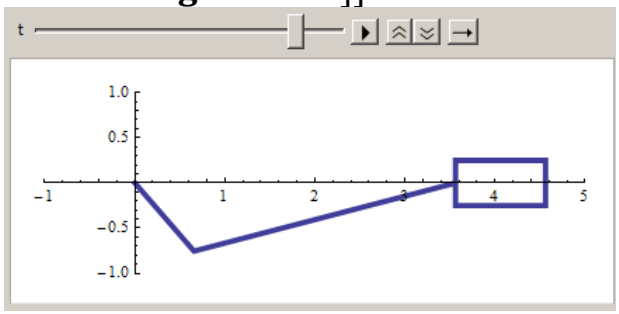
$$\mathbf{Animate[ParametricPlot3D[}$$
$$\mathbf{\{6Cos[u](1 + Sin[u]) + 4(1 - \frac{Cos[u]}{2})Cos[\frac{u}{2} + \frac{\pi}{4}]Cos[v],}$$
$$\mathbf{16Sin[u] + 4(1 - \frac{Cos[u]}{2})Sin[\frac{u}{2} + \frac{\pi}{4}]Cos[v], 4(1 - \frac{Cos[u]}{2})Sin[v]\}}$$
$$\mathbf{\{u, 0, U\}, \{v, 0, 2\pi\}, PlotRange \to \{\{-11, 10\}, \{-20, 20\}, \{-6, 6\}\},}$$
$$\mathbf{PlotPoints \to 30, Mesh \to None, ImageSize \to 280],}$$
$$\mathbf{\{U, 0.01, 2\pi\}, AnimationRunning \to False]}$$



2.4.2 Моделирование механических движений

Пример 1. Кривошипно – шатунный механизм, является устройством, которое вращательное движение преобразует в поступательное или наоборот. Смоделируем схематически движение этого механизма, изобразив его в виде ломаной.

```
Clear[xl, yl, pt];  w = 1;
xll = {0, Cos[wt], Sqrt[9 - Sin[wt]^2], 0, 1, 0, -1, 0};
yll = {0, Sin[wt], -Sin[wt], -1/4, 0, 1/2, 0, -1/4};
xl = Accumulate[xll];
yl = Accumulate[yll];
DynamicModule[{pt},
  pt[t_] = Transpose[{xl, yl}];
  Animate[ListLinePlot[pt[t],
      AspectRatio → Automatic, PlotRange → {{-1, 5}, {-1, 1}},
      PlotStyle → Thickness[0.01]],
  {t, 0, 2π}, AnimationRunning → False]]
```



Здесь списки **xll** и **yll** содержат длины проекций отрезков ломаной, представляющей механизм, на оси X и Y соответственно. Функция `Accumulate[List]` возвращает список накопительных сумм элементов списка `List`. Например,

```
Accumulate[{a, b, c, d}]
```

$\{a, a+b, a+b+c, a+b+c+d\}$

В результате списки **xl** и **yl** содержат x и y координаты узлов ломаной. Из них создается список **pt[t]** пар координат $\{\ldots,\{x_i, y_i\},\ldots\}$ узлов, которые используются функцией `ListLinePlot` для рисования ломаной.

Пример 2. Поперечные колебания невесомой струны с грузами.

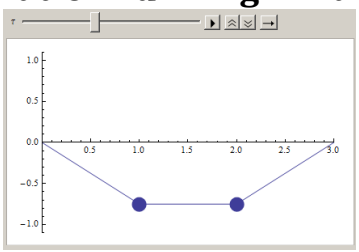
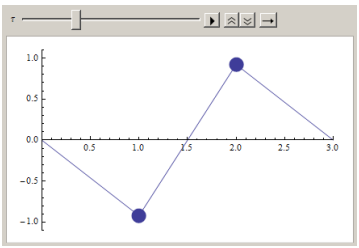
Рассмотрим колебания механической системы, которая состоит из двух частиц, каждая массой `M`, закрепленных на невесомой упругой струне. Массы разбивают струну на три участка одинаковой длины. Через y_1 обозначим вертикальное смещение 1-й массы и через y_2 – смещение 2-й массы. Известно, что колебание системы может быть представлено в виде [1]

$$y_1(t) = A_1 \cos(w_1 t + \varphi_1) + A_2 \cos(w_2 t + \varphi_2),$$
$$y_2(t) = A_1 \cos(w_1 t + \varphi_1) - A_2 \cos(w_2 t + \varphi_2),$$

где значения постоянных определяется из начальных условий и механических характеристик системы.

Рассмотрим колебание системы без начальной скорости (начальные скорости частиц равны нулю). Длина струны равна 3. Колебание, при котором массы двигаются синхронно, называется первой модой. Следующий код моделирует первую моду колебаний (следующий рисунок слева).

```
DynamicModule[{w1 = 1, y1, y2, ptpt, pt},
  y1[t_] = Cos[w1t];
  y2[t_] = Cos[w1t];
  Animate[
    ptpt = {{1, y1[t]}, {2, y2[t]}};
    pt = {{0, 0}, {1, y1[t]}, {2, y2[t]}, {3, 0}};
    p1 = ListLinePlot[pt/. t → τ, PlotRange → {{0, 3}, {−1.1, 1.1}}];
    p2 = ListPlot[ptpt/. t → τ, PlotStyle → PointSize[0.05]];
    Show[p1, p2],
  {τ, 0, 6}, AnimationRunning → False]]
```

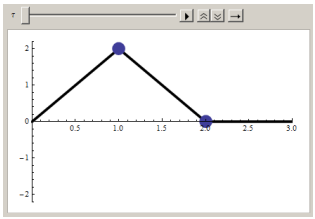
Вторая мода колебаний представляет движение частиц в противофазе. Чтобы построить такое движение в предыдущем коде мы должны изменить знак у функции y_2 и, возможно, изменить угловую частоту. Первые три строки предыдущего кода примут новый вид

```
DynamicModule[{w2 = √3, y1, y2, ptpt, pt},
  y1[t_] = Cos[w2t];
  y2[t_] = −Cos[w2t];
  …
]
```

Остальные строки кода остаются без изменений. В результате получается движение, представленное в панели анимации предыдущего рисунка справа.

Смешанное колебание состоит из суперпозиции обеих мод, вклад каждой из которых определяется начальным положением масс системы. Построим колебание, начальное положение масс которого показано на следующем рисунке. Для этого изменим первые три строки кода следующим образом

```
DynamicModule[{w1 = 1, w2 = √3, y1, y2, ptpt, pt},
  y1[t_] = Cos[w1t] + Cos[w2t];
  y2[t_] = Cos[w1t] − Cos[w2t];
  …
]
```



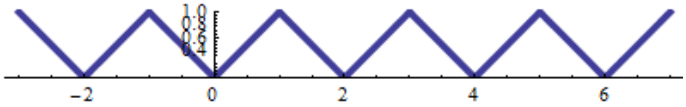
Пример 3. *Математический бильярд в прямоугольнике*. Имеется горизонтальный бильярдный стол произвольной формы без луз (плоская область). По этому столу без трения движется точечный шар, упруго отражаясь от бортов. Требуется определить траекторию этого шара/точки. Такая механическая система – точечный шар в области Q, ограниченной бортом Г (границей области Q) – называется математическим бильярдом. Траектория точки в математическом бильярде определяется ее начальным положением и начальным вектором скорости.

Введем вспомогательную функция $stc(x,w)$, которая будет обозначать пилообразную непрерывную (**s**aw **t**ooth **c**ontinuous) кусочно-линейную функцию, определяемую, например, уравнением

$$stx(x,w)=\left|x-w\left[\frac{x}{w}+\frac{1}{2}\right]\right| \quad \text{или} \quad stc(x,w)=\frac{w}{2\pi}\arccos\cos\frac{2\pi x}{w},$$

где $[z]$ (квадратные скобки) является функцией взятия наибольшего целого, не превосходящего z.

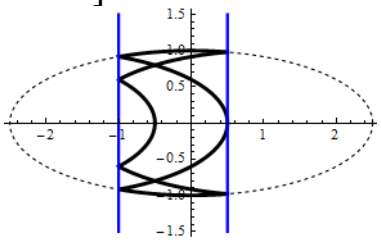
```
Stc[x_, w_] = Abs[x - w Floor[x/w + 1/2]];
Plot[Stc[x, 2], {x, -3, 7}, , AspectRatio → Automatic]
```



Приведем без доказательства некоторые факты, связанные с преобразованием отражения кривых.

Пусть кривая С задана параметрическими уравнениями $x=x(t),\ y=y(t)$. Уравнение кривой C_t, полученной многократным внутренним отражением от пары вертикальных прямых $x=x_l$, $x=x_r$ $(x_l<x_r)$, имеет вид $x_t=x_l+stc\left(x(t)-x_l,2(x_r-x_l)\right),\ y_t=y(t)$. Вот пример кривой, полученной отражением эллипса от двух вертикальных прямых.

```
xl = -1; xr = 1/2;
x[t_] = 2.5Cos[t];
y[t_] = Sin[t];
xt[t_] = xl + Stc[x[t] - xl, 2(xr - xl)];
yt[t_] = y[t];
pt = ParametricPlot[{xt[t], yt[t]}, {t, 0, 2π}];
pe = ParametricPlot[{x[t], y[t]}, {t, 0, 2π}];
pl = Graphics[{Blue, Thick,
      {Line[{{xl, -1.5}, {xl, 1.5}}], Line[{{xr, -1.5}, {xr, 1.5}}]}}];
Show[pe, pt, pl, PlotRange → All]
```



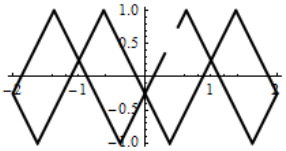
Аналогично, уравнение кривой C_t, полученной многократным внутренним отражением C от пары горизонтальных прямых $y = y_d$, $y = y_u$ $(y_d < y_u)$ имеет вид $x_t = x(t),\ y_t = y_d + stc(y(t) - y_d, 2(y_u - y_d))$.

Параметрическое уравнение кривой C_t, полученной многократным внутренним отражением C от границ прямоугольника $x_l \le x \le x_r,\ y_d \le y \le y_u$, имеет вид

$$x_t(t) = x_l + stc(x(t) - x_l, 2(x_r - x_l))$$
$$y_t(t) = y_d + stc(y(t) - y_d, 2(y_u - y_d)) \quad (1)$$

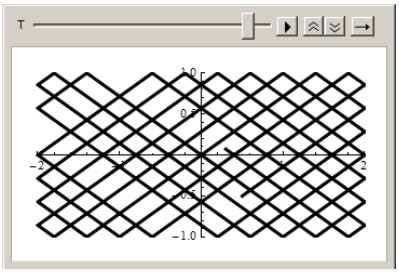
Вот пример уравнений ломаной, полученной многократным внутренним отражением прямой/луча от границ прямоугольника $-2 \le x \le 2, -1 \le y \le 1$. Луч стартует в точке $(0.5, 0.75)$ и имеет направляющий вектор $(1, 2)$.

```
xl = -2; xr = 2; yd = -1; yu = 1;
x[t_] = 0.5 + t;
y[t_] = 0.75 + 2t;
xt[t_] = xl + Stc[x[t] - xl, 2(xr - xl)];
yt[t_] = yd + Stc[y[t] - yd, 2(yu - yd)];
ParametricPlot[{xt[t], yt[t]}, {t, 0, 7.8},
        PlotStyle → {Black, Thickness[0.01]}]
```



Траектория точки в прямоугольном бильярде является ломаной, получаемой многократным отражением луча от «бортов» прямоугольника. Пусть из точки (x_0, y_0) вылетает бесконечно малый шар в направлении вектора $(\cos\alpha, \sin\alpha)$. Трение отсутствует. Если бы бортов области не было, то он двигался бы вдоль прямой с уравнением $x = x_0 + c\,t\cos\alpha$, $y = y_0 + c\,t\sin\alpha$, где t – время, c – скорость, α – угол наклона луча. Уравнение бильярдной траектории будет иметь вид (1), где в качестве функций $x(t), y(t)$ следует использовать уравнение этой прямой. Следующий код моделирует движение точки в прямоугольном бильярде.

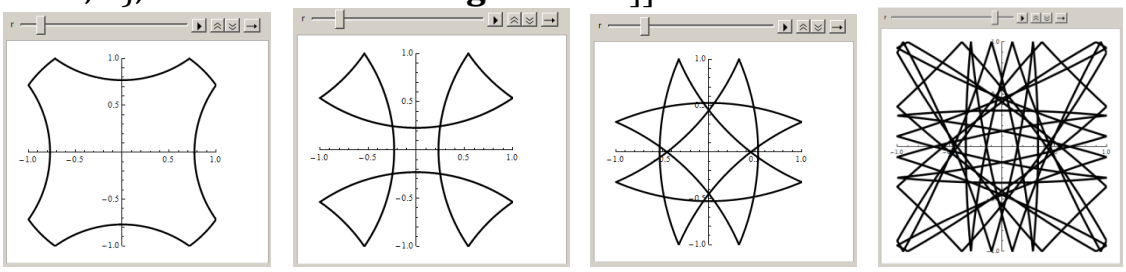
```
DynamicModule[{xl = -2, xr = 2, yd = -1, yu = 1,
        x0 = 0.5, y0 = -0.5, α = ArcTan[5/7], x, y, xt, yt},
  x[t_] = x0 + t Cos[α];
  y[t_] = y0 + t Sin[α];
  xt[t_] = xl + Stc[x[t] - xl, 2(xr - xl)];
  yt[t_] = yd + Stc[y[t] - yd, 2(yu - yd)];
  Animate[ParametricPlot[{xt[t], yt[t]}, {t, 0, T},
          PlotStyle → {Black, Thickness[0.01]},
          PlotRange → {{xl, xr}, {yd, yu}}, PlotPoints → 200],
  {T, 0.01, 68}, AnimationRunning → False]]
```



Пример 4. *Отражение кругового волнового фронта*.

Формулы отражения (1) кривой $x = x(t), y = y(t)$ от «бортов» прямоугольника можно использовать для кривой любой формы (не только прямой). Например, уравнение кругового волнового фронта распространяющегося из некоторой точки и отражающегося от «берегов» прямоугольника, тоже будут преобразовываться в соответствии с формулами (1). В следующем примере мы строим анимацию распространения и отражения кругового волнового фронта в квадратной области. Начальная точка расположена в центре.

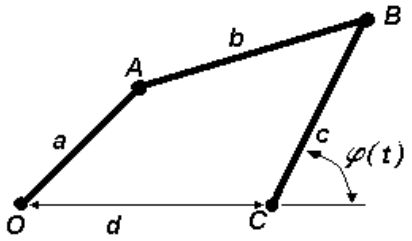
```
DynamicModule[{xl = -1, xr = 1, yd = -1, yu = 1, x, y, xt, yt},
  x[r_] = r Cos[a];
  y[r_] = r Sin[a];
  xt[r_] = xl + Stc[x[r] - xl, 2(xr - xl)];
  yt[r_] = yd + Stc[y[r] - yd, 2(yu - yd)];
  Animate[ParametricPlot[{xt[r], yt[r]}, {a, 0, 2π},
                PlotStyle → {Black, Thickness[0.01]},
                PlotRange → {{xl, xr}, {yd, yu}}, PlotPoints → 200],
  {r, 0.01, 6}, AnimationRunning → False]]
```



Несколько кадров волнового фронта в последовательные моменты времени представлены на предыдущем рисунке.

Пример 5. Моделирование кривых с помощью шарнирных механизмов

Смоделируем движение шарнирного механизма, состоящего из трех звеньев. Ведущее звено OA вращается вокруг точки O с постоянной угловой скоростью, звенья OA, AB и BC шарнирно соединены в точках A и B. Звено BC также может совершать свободное вращение вокруг точки C. Обозначим длины звеньев через *a*, *b*, *c* так, как показано на рисунке и расстояние по горизонтали между точками O и C обозначим через *d*.



Чтобы точка A могла пройти горизонтальное положение слева от точки O необходимо, чтобы выполнялось соотношение $a+d <= b+c$.

Очевидно, что точка B будет двигаться по дуге окружности вокруг точки C, однако закон этого движения нам неизвестен. Обозначим угол, который образует отрезок CB с осью X через φ(t). Тогда

$$\begin{cases} x_A = a\cos t \\ y_A = a\sin t \end{cases} \qquad \begin{cases} x_B = d + c\cos\varphi(t) \\ y_B = c\sin\varphi(t) \end{cases}$$

Длина отрезка AB равна $(x_A - x_B)^2 + (y_A - y_B)^2 = b^2$. Подставим сюда выражения координат точек A и B из предыдущих соотношений и получим уравнение для определения неизвестной функции φ(t).

$$(d + c\cos\varphi(t) - a\cos t)^2 + (c\sin\varphi(t) - a\sin t)^2 = b^2.$$

После преобразований получаем $\alpha\cos\varphi(t) - \beta\sin\varphi(t) = \gamma$, где

$$\alpha = d - a\cos t, \quad \beta = a\sin t, \quad \gamma = \frac{b^2 - a^2 - c^2 - d^2 + 2ad\cos t}{2c}$$

Разделим обе части последнего уравнения на $\sqrt{\alpha^2 + \beta^2}$ и обозначим

$$\sin\delta = \frac{\alpha}{\sqrt{\alpha^2 + \beta^2}}, \quad \cos\delta = \frac{\beta}{\sqrt{\alpha^2 + \beta^2}}$$

Тогда уравнение относительно φ(t) может быть записано в виде $\sin(\delta - \varphi(t)) = \frac{\gamma}{\sqrt{\alpha^2 + \beta^2}}$ или $\varphi(t) = \delta(t) - \arcsin\frac{\gamma}{\sqrt{\alpha^2 + \beta^2}}$. Учитывая, что имеет место равенство $\alpha^2 + \beta^2 = d^2 - 2ad\cos t + a^2$, после преобразований получаем

$$\varphi(t) = \delta(t) - \zeta(t),$$

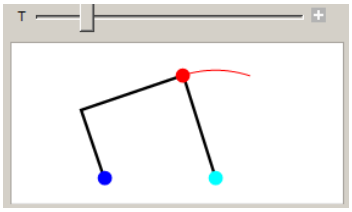
где

$$\delta(t) = \arccos\frac{a\sin t}{\sqrt{d^2 - 2ad\cos t + a^2}} \quad \text{и} \quad \zeta(t) = \arcsin\frac{b^2 - a^2 - c^2 - d^2 + 2ad\cos t}{2c\sqrt{d^2 - 2ad\cos t + a^2}}.$$

Следующий код моделирует движение трехзвенного механизма

```
DynamicModule[{a = 2, b = 3, c = 3, d = 3.1, δ, ζ, φ, ln, pw, pc, pts},
 δ[t_] = ArcCos[a Sin[t] / Sqrt[a^2 + d^2 - 2 a d Cos[t]]];
 ζ[t_] = ArcSin[(b^2 + 2 a d Cos[t] - a^2 - c^2 - d^2) / (2 c Sqrt[a^2 + d^2 - 2 a d Cos[t]])];
 φ[t_] = δ[t] - ζ[t];
```

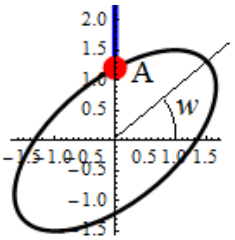
```
Manipulate[
  ln = Line[{{0, 0}, {aCos[t], aSin[t]}, {d + cCos[φ[t]], cSin[φ[t]]}, {d, 0}}];
  pts = Graphics[{{Blue, Disk[{0, 0}, 0.2]},
                  {Red, Disk[{d + cCos[φ[T]], cSin[φ[T]]}, 0.2]},
                  {Cyan, Disk[{d, 0}, 0.2]}}];
  pw = Graphics[{Thickness[0.01], Black, ln /. t → T}];
  pc = ParametricPlot[{d + cCos[φ[t]], cSin[φ[t]]},
        {t, π/4 + 0.001, T}, PlotStyle → Red];
  Show[pw, pc, pts, PlotRange → {{-a, c + d}, {-c - 0.2, c + 0.2}}],
{T, π/4, 2π + π/4}]
```

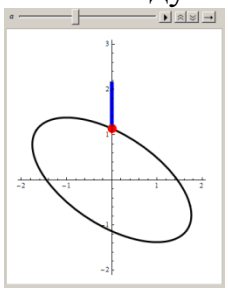


Здесь мы использовали панель манипулятора, а не панель анимации, как в предыдущих примерах.

Пример 6. Кулачковый механизм. Эксцентрик в форме эллипса совершает вращательное движение и взаимодействует с кулачком в форме прямолинейного стержня. Смоделируем движение такого механизма, используя графические функции системы.

Пусть полуоси эллипса равны a и b, а длина кулачка/стержня L. В следующем коде функции $xe(t,w)$, $ye(t,w)$ при фиксированном w представляют параметрическое уравнение повернутого на угол w эллипса. Функция $y_A(w)$ представляет y - координату толкателя (точка A на следующем рисунке слева).





```
DynamicModule[{a = 2, b = 1, L = 1, xe, ye, yA, ys},
  xe[t_, w_] = aCos[t]Cos[w] - bSin[t]Sin[w];
  ye[t_, w_] = aCos[t]Sin[w] + bSin[t]Cos[w];
  yA[w_] = a b / Sqrt[a^2 Cos[w]^2 + b^2 Sin[w]^2];
  ys[t_, w_] = yA[w] + t;
  Animate[
    p1 = ParametricPlot[{xe[t, α], ye[t, α]}, {t, 0, 2π},
                        PlotStyle → {Black, Thickness[0.01]}];
    p2 = ParametricPlot[{0, ys[t, α]}, {t, 0, L},
                        PlotStyle → {Blue, Thickness[0.02]}];
    pt = Graphics[{Red, Disk[{0, yA[α]}, 0.1]}];
```

```
    Show[p1, p2, pt, PlotRange → {{-a, a}, {-a, a + 1}}],
  {α, 0, 2π}, AnimationRunning → False]]
```

Панель анимации со схематическим изображением кулачкового механизма показана на предыдущем рисунке справа.

Параметрическое уравнение повернутого эллипса $xe(t,w)$, $ye(t,w)$ получается умножением вектора $(a\cos t, b\sin t)$ на матрицу поворота на угол *w*. Для определения y координаты точки А мы определяем значение параметра *t* в точке А на повернутом эллипсе. Для этого приравниваем нулю функцию $xe(t,w)$ и выражаем *t* через *w* (точнее выражаем $tg\,t$). Получаем $tg\,t = \dfrac{a\cos w}{b\sin w}$.

Подстановка этого значения в выражение $ye(t,w)$ после упрощений дает функцию $y_A(w)$.

2.4.3 Движение жидкости в трубах

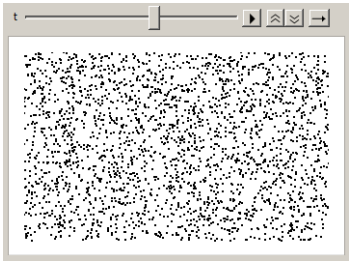
Смоделируем поле скоростей жидкости в цилиндрической трубе с эллиптическим сечением. Скорость жидкости w направлена вдоль оси трубы (ось X), одинакова во всех поперечных сечениях, и в точке сечения с координатами (y,z) определяется формулой

$$w(y,z) = \frac{Ca^2b^2}{a^2+b^2}\left(1-\frac{y^2}{a^2}-\frac{z^2}{b^2}\right),$$

где *a* и *b* являются длинами полуосей эллипса/сечения, C – некоторая постоянная, зависящая от разности давлений на концах трубы и вязкости жидкости.

Пример 3.1. Смоделируем движение жидкости как движение частиц в продольном сечении z=0 трубы. Ось X направлена вдоль оси трубы вправо, ось Y – вертикально вверх. Положим $a=1,\ b=1,\ C=2$. Сгенерируем большое количество случайно расположенных в продольном сечении точек и, в зависимости от их вертикальной координаты *y*, зададим им скорость движения $w(y)=1-y^2$.

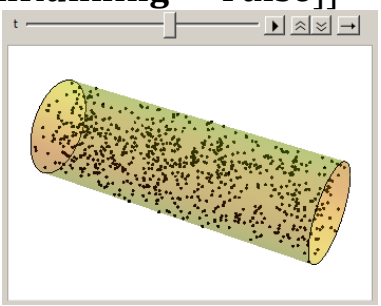
```
DynamicModule[{n = 5000, h = 6, w, xl, yl, pt},
  w[y_] = 1 - y^2;
  xl = RandomReal[{-h, h}, n];
  yl = RandomReal[{-1, 1}, n];
  Animate[
    pt = Transpose[{xl + w[yl]t, yl}];
    ListPlot[pt, PlotStyle → {Black, PointSize[0.005]},
                 PlotRange → {{0, h}, {-1, 1}}, Axes → None],
  {t, 0, h, 0.01}, AnimationRunning → False]]
```



Отметим, что приведенное здесь статическое изображение, не передает процесс движения в отличие от динамического движения, воспроизводимого в панели анимации.

Пример 3.2. Смоделируем пространственное движение жидкости как движение частиц в круглой трубе. Ось X направлена вдоль оси трубы вправо. Положим $a=1,\ b=1,\ C=2$. Сгенерируем большое количество случайно расположенных в трубе точек и, в зависимости от их поперечных координат *y* и *z*, зададим им скорость движения $w(y,z)=1-y^2-z^2$.

```
DynamicModule[{n = 2000, h = 6, w, xl, yl, zl, pt},
  w[y_, z_] = 1 - y^2 - z^2;
  xl = RandomReal[{-h, h}, n];
  yl = RandomReal[{-1, 1}, n];
  zl = RandomReal[{-1, 1}, n];
  Animate[
    pt = Transpose[{xl + w[yl, zl] t, yl, zl}];
    pl = ListPointPlot3D[pt, PlotStyle → {Black, PointSize[0.005]},
                  PlotRange → {{0, h}, {-1, 1}, {-1, 1}},
                  RegionFunction → Function[{x, y, z}, y^2 + z^2 < 1]];
    pr = RegionPlot3D[y^2 + z^2 < 1, {x, 0, h}, {y, -1, 1}, {z, -1, 1},
          PlotStyle → Directive[Yellow, Opacity[0.3]], Mesh → None];
    Show[pl, pr, BoxRatios → Automatic, Boxed → False, Axes → None],
  {t, 0, h, 0.001}, AnimationRunning → False]]
```



Особенностью этого примера является создание полупрозрачной трехмерной области **pr**, представляющей участок трубы, и использование у функции `ListPointPlot3D` опции `RegionFunction`, отсекающей изображения точек, выходящие за пределы кругового сечения трубы.

□

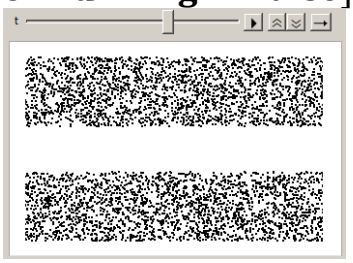
Поле скоростей жидкости в области между двумя коаксиальными круглыми цилиндрами радиусов r_1 и r_2 $(r_2 > r_1)$ определяется по формуле

$$w(r) = C\left(r_1^2 - r^2 + \frac{\left(r_2^2 - a^2\right)\ln\left(r / r_1\right)}{\ln\left(r_2 / r_1\right)} \right),$$

где *r* – расстояние точки сечения до оси и C – некоторая постоянная, зависящая от вязкости жидкости и разности давлений на входе и выходе трубы.

Пример 3.3. Смоделируем движение жидкости в продольном сечении z=0 такой трубы. Ось X направим вдоль оси трубы вправо и положим $r_1 = 0.5,\ r_2 = 2,\ C = 1$. Сгенерируем большое количество случайно расположенных в продольном сечении точек и, в зависимости от их расстояния *r* до оси, зададим им скорость движения $w(r)$.

```
DynamicModule[{n = 5000, h = 6, r1 = 0.5, r2 = 2, w, xl, yl1, yl2, pt},
  w[r_] = r1^2 - r^2 + (r2^2 - r1^2)Log[r/r1]/Log[r2/r1];
  xl = RandomReal[{-h, h}, n];
  yl1 = RandomReal[{r1, r2}, n];
  yl2 = RandomReal[{-r2, -r1}, n];
  Animate[
    pt = Join[Transpose[{xl + w[Abs[yl1]]t, yl1}],
              Transpose[{xl + w[Abs[yl2]]t, yl2}]];
    ListPlot[pt, PlotStyle-> {Black, PointSize[0.005]},
             PlotRange → {{0, h}, {-2, 2}}, Axes → None],
  {t, 0, h - 1, 0.01}, AnimationRunning → False]]
```

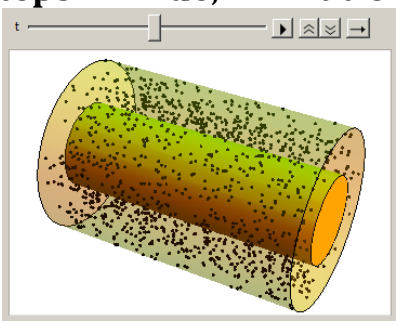


В этом примере мы создаем два случайных числовых множеств **yl1** и **yl2**, которые затем используются при создании единого списка **pt** пар координат точек.

Пример 3.4. Смоделируем пространственное движение жидкости в области между двумя коаксиальными круглыми цилиндрами. Ось X направим вдоль оси цилиндров и положим $r_1 = 1,\ r_2 = 2,\ C = 1$. Сгенерируем большое количество случайно расположенных в трубе точек и, в зависимости от их расстояния *r* до оси, зададим им скорость движения $w(r)$.

```
DynamicModule[{n = 3000, h = 6, r1 = 1, r2 = 2, w, rl, φl, xl, yl, zl, pt},
  w[r_] = r1^2 - r^2 + (r2^2 - r1^2)Log[r/r1]/Log[r2/r1];
  xl = RandomReal[{-h, h}, n];
  rl = RandomReal[{r1, r2}, n];
  φl = RandomReal[{0, 2π}, n];
  yl = rl Cos[φl];
  zl = rl Sin[φl];
```

```
Animate[
  pt = Transpose[{xl + w[rl] t, yl, zl}];
  pl = ListPointPlot3D[pt, PlotStyle -> {Black, PointSize[0.005]},
                PlotRange → {{0, h}, {-r2, r2}, {-r2, r2}},
                RegionFunction → Function[{x, y, z},
                      y^2 + z^2 > r1^2 && y^2 + z^2 < r2^2]];
  pr1 = RegionPlot3D[y^2 + z^2 < r2^2, {x, 0, h}, {y, -r2, r2}, {z, -r2, r2},
          PlotStyle → Directive[Yellow, Opacity[0.3]], Mesh → None];
  pr2 = RegionPlot3D[y^2 + z^2 < r1^2, {x, 0, h}, {y, -r2, r2}, {z, -r2, r2},
          PlotStyle → Directive[Yellow, Opacity[1]], Mesh → None];
  Show[pl, pr1, pr2, BoxRatios → Automatic, Boxed → False,
                                          Axes → None],
{t, 0, h, 0.1}, AnimationRunning → False,
          DisplayAllSteps → True, AnimationRate → 2]]
```



В этом примере мы создаем полупрозрачный внешний цилиндр и непрозрачный внутренний. Координаты случайных точек создаются путем генерирования множества **rl** случайных радиусов в диапазоне от r1 до r2 и множества **φl** случайных полярных углов, которые затем используются для создания списка **pt** координат $\{...,\{x_i, y_i, z_i\},...\}$ точек.

2.4.4 Двумерные волновые движения

Пример 4.1. Поверхностные волны Релея. Английский физик Релей показал, что в упругом полупространстве могут существовать специальные волны, которые он назвал поверхностными. У таких волн амплитуда колебаний быстро уменьшается с удалением от поверхности. Одно из таких плоских волновых движений может быть описано с помощью следующих функций [9]

$$U(x,y,t) = C\left(-e^{-\alpha y} + c\,e^{-\beta y}\right)\sin\left(\xi x + p t\right)$$
$$V(x,y,t) = C\,a\left(-e^{-\alpha y} - d\,e^{-\beta y}\right)\cos\left(\xi x + p t\right) \qquad (1)$$

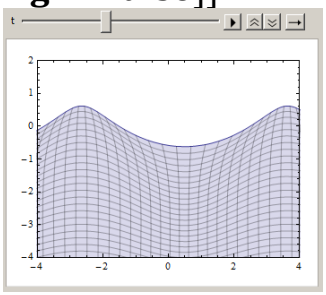
где U и V представляют продольное и поперечное смещение точек среды (x, y) относительно положения равновесия. Здесь $\alpha = a\xi$, $\beta = b\xi$ и a, b, c, d некоторые постоянные, зависящие от свойств среды, C – произвольная постоянная. Ось Y направлена в глубь среды.

Направим ось Y вертикально вверх (у системы *Mathematica* нет простого способа менять направления осей). В силу этого в примере в формулах Релея (1) мы изменим знак перед переменной y и перед функцией V. Полупространство в состоянии покоя будем моделировать в виде области,

заданной в параметрическом виде $x(u,v)=u,\ y(u,v)=v$, где $-\infty<u<\infty, v\le 0$. Тогда смещенные точки области в момент времени t будут иметь координаты $X=x(u,v)+U(x(u,v),y(u,v),t),\ Y=y(u,v)+V(x(u,v),y(u,v),t)$, где $v\le 0$.

Следующий код моделирует движение среды – поверхностную волну Релея.

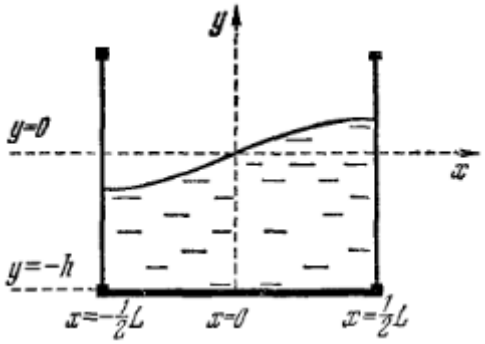
```
DynamicModule[{a = 0.8475, b = 0.3933,
               c = 0.5753, d = 1.732, p = 1, Cc = 1, U, V, X, Y},
  U[x_, y_, t_] = Cc(-Exp[-a y] + c Exp[-b y]) Sin[x + p t];
  V[x_, y_, t_] = Cc a (Exp[-a y] - d Exp[-b y]) Cos[x + p t];
  X[u_, v_, t_] = u + U[u, -v, t];
  Y[u_, v_, t_] = v - V[u, -v, t];
  Animate[ParametricPlot[
    {{X[u, v, t], Y[u, v, t]}}, {u, -5, 5}, {v, -5, 2}, Mesh → {31, 26},
     RegionFunction → Function[{x, y, u, v}, v < 0],
     PlotRange → {{-4, 4}, {-4, 2}}, Axes → None],
  {t, 0, 2π}, AnimationRunning → False]]
```



Использование параметрического уравнения области в этом примере позволяет показать не только форму поверхности, но и образ декартовой координатной сетки внутри среды.

Пример 4.2. Установившаяся синусоидальная волна в прямоугольном аквариуме постоянной глубины [1].

Рассмотрим прямоугольный аквариум шириной L с жидкостью налитой до уровня h. Направим ось Y вертикально вверх с началом на свободной поверхности жидкости, а ось X направим вправо (см. следующий рисунок).



Одно из возможных установившихся волновых движений жидкости может быть представлено в виде

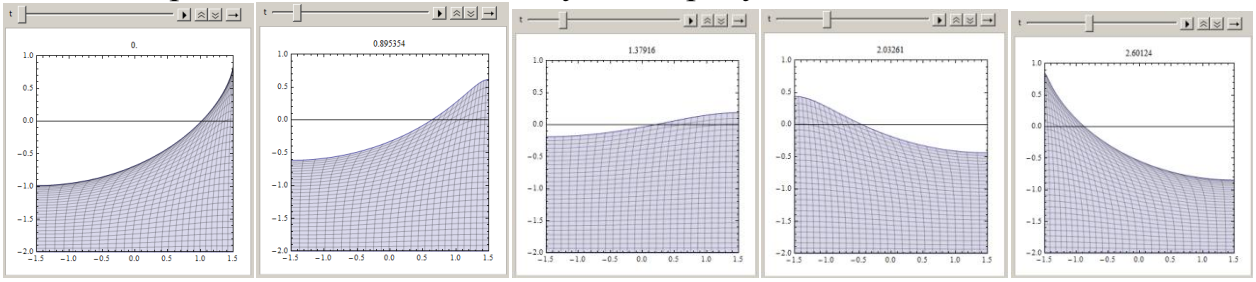
$$\begin{aligned} U &= a\cos(wt)\cos(k\,x)\left(e^{k\,y}+e^{-2k\,h}e^{-k\,y}\right)\\ V &= a\cos(wt)\sin(k\,x)\left(e^{k\,y}-e^{-2k\,h}e^{-k\,y}\right)\end{aligned} \qquad (2)$$

где $k=\pi/L$ и a – произвольная постоянная. Функции $U(x,y,t)$ и $V(x,y,t)$ дают мгновенные значения смещений частицы жидкости с равновесными координатами x, y в направлениях осей X и Y.

Жидкость в состоянии покоя будем моделировать в виде плоской области, заданной в параметрическом виде $x(u,v)=u$, $y(u,v)=v$, где $-L/2\le u\le L/2, -h\le v\le 0$. Точки области в момент времени t будут иметь координаты $X=x(u,v)+U(x(u,v),y(u,v),t)$, $Y=y(u,v)+V(x(u,v),y(u,v),t)$, где надо положить $x=u$, $y=v$ и $-L/2\le u\le L/2, -h\le v\le 0$.

```
DynamicModule[{w = 1, L = 3, a = 1, h = 2, k, X, Y}, k = π/L;
  X[u_, v_, t_] = u + aCos[wt]Cos[ku](Exp[kv] + Exp[-2kh]Exp[-kv]);
  Y[u_, v_, t_] = v + aCos[wt]Sin[ku](Exp[kv] - Exp[-2kh]Exp[-kv]);
  Animate[
    ParametricPlot[{{X[u, v, t], Y[u, v, t]}}, {u, -L/2, L/2}, {v, -h, 0},
        Mesh → 26, PlotRange → {{-L/2, L/2}, {-h, 1.}},
        PlotLabel → t, Axes → {True, False}],
  {t, 0, 2π}, AnimationRunning → False]]
```

Область, занимаемая жидкостью, в начальный и некоторые последующие моменты времени показана на следующем рисунке.

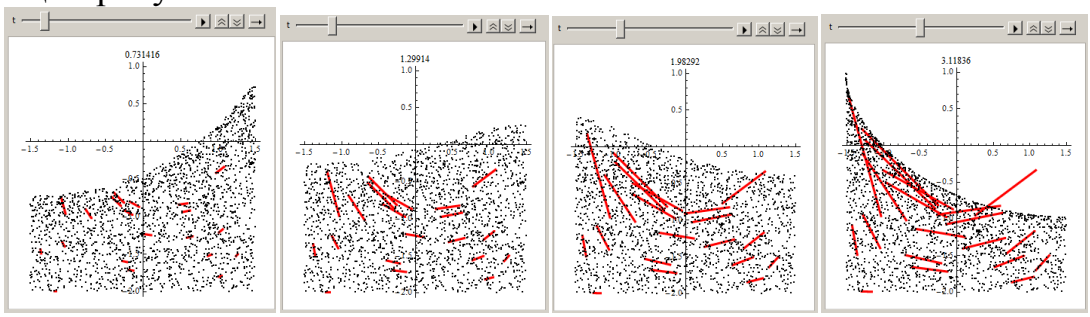


Наблюдая анимацию, можно заметить, что у стенок и дна движение происходит вдоль границ. Из уравнений (2) видно, что движение любой фиксированной точки (x,y) состоит из гармонического колебания по некоторому прямолинейному отрезку в плоскости XY. Этот факт можно продемонстрировать графически, если среду смоделировать в виде множества точек и нарисовать траектории некоторых из них.

```
DynamicModule[{n = 2000, L = 3, h = 2, a = 1, w = 1, np = 20,
                k, ul, vl, X, Y, nl, up, vp, Xp, Yp, Xl, Yl, pt, pp, pl},
  ul = RandomReal[{-L/2, L/2}, n];
  vl = RandomReal[{-h, 0}, n];
  k = π/L;
  X[u_, v_, t_] = u + aCos[wt]Cos[ku](Exp[kv] + Exp[-2kh]Exp[-kv]);
  Y[u_, v_, t_] = v + aCos[wt]Sin[ku](Exp[kv] - Exp[-2kh]Exp[-kv]);
  SeedRandom[1234]; (* инициализация генератора случайных чисел *)
  nl = RandomInteger[{1, n}, np];
  up = ul[[nl]];   vp = vl[[nl]];
  Xp[τ_] = X[up, vp, τ];   Yp[τ_] = Y[up, vp, τ];
```

```
Animate[
  Xl = X[ul, vl, t];
  Yl = Y[ul, vl, t];
  pt = Transpose[{Xl, Yl}];
  pp = ParametricPlot[Transpose[{Xp[τ], Yp[τ]}], {τ, 0, t},
                      PlotStyle → {Red, Thickness[0.01]}];
  pl = ListPlot[pt, PlotStyle → {Black, PointSize[0.01]},
                      PlotRange → {{-L/2, L/2}, {-h, 1.}}];
  Show[pp, pl, PlotRange → {{-L/2, L/2}, {-h, 1.}},
                      AxesOrigin → Automatic],
 {t, 0.01, 2π}, AnimationRunning → False]]
```

Положение точек жидкости в последовательные моменты времени показано на следующем рисунке.



В этом коде из `n` случайных точек выбираются `np`. Для них строятся траектории движения, которые, как видно из рисунков, являются отрезками.

Пример 4.3. Установившееся волновое движение жидкости в резервуаре конечной ширины и бесконечной глубины [1].

Рассмотрим сосуд шириной L бесконечной глубины. Ось X направлена вправо, ось Y вверх. Начало координат расположено в середине свободной поверхности жидкости. Одно из возможных установившихся волновых движений жидкости может быть получено из формул (2) при $h = \infty$.

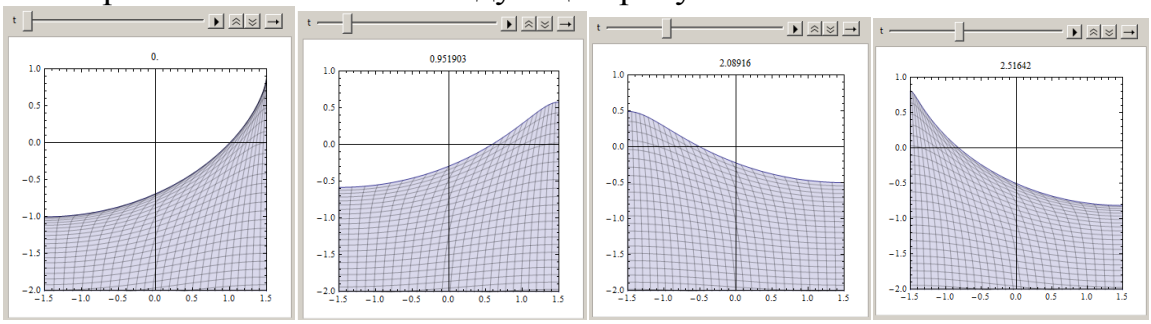
$$\begin{aligned} X &= x + a\cos(wt)\cos(k\,x)\,e^{k\,y} \\ Y &= y + a\cos(wt)\sin(k\,x)\,e^{k\,y} \end{aligned} \qquad (-L/2 < u < L/2, v \le 0), \qquad (3)$$

где (x,y) координаты точек среды в состоянии покоя, X и Y координаты тех же точек в момент времени t, и a, w, k – некоторые постоянные.

Для создания анимации аналогично предыдущему примеру, жидкость в состоянии покоя будем моделировать в виде плоской области, заданной в параметрическом виде $x(u,v) = u$, $y(u,v) = v$, где $-L/2 < u < L/2, v \le 0$. Положение точек области в движении будут определяться формулами (3).

```
DynamicModule[{w = 1, L = 3, k = π/L, a = 1, X, Y},
  X[u_, v_, t_] = u + aCos[wt]Cos[ku]Exp[kv];
  Y[u_, v_, t_] = v + aCos[wt]Sin[ku]Exp[kv];
  Animate[ParametricPlot[
          {{X[u, v, t], Y[u, v, t]}}, {u, -L/2, L/2}, {v, -2.5, 0}, Mesh → 26,
            PlotRange → {{-L/2, L/2}, {-2, 1}}, PlotLabel → t],
  {t, 0, 2π}, AnimationRunning → False]]
```

Область, занимаемая жидкостью, в начальный и некоторые последующие моменты времени показана на следующем рисунке.



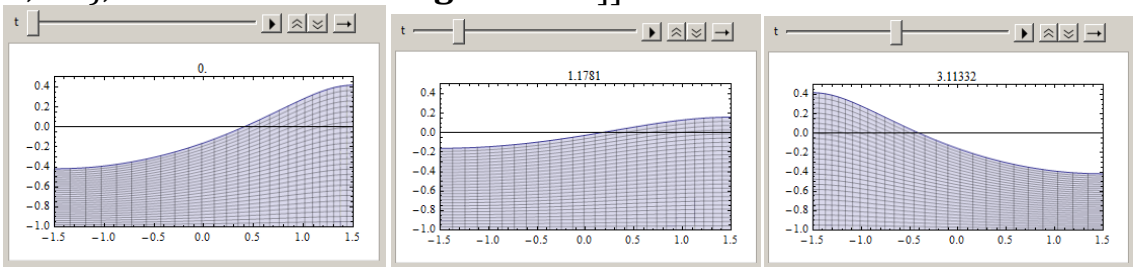
Здесь, как и в предыдущем примере, частицы жидкости двигаются по прямолинейным отрезкам.

Пример 4.4. Стоячие волны в неглубокой воде [1]. Под ними понимают волны, которые возникают в сосуде, равновесная глубина h которого мала по сравнению с длиной волн $\lambda = 2\pi / k$. В этом случае мы можем аппроксимировать зависимости U и V от y, оставив в разложении в ряд Тейлора только первые члены. В результате уравнения (2) примут вид

$$U = 2a\cos(wt)\cos(k\,x)$$
$$V = 2a\cos(wt)\sin(k\,x)\,(k(y+h))$$

Для такой волны горизонтальное смещение частиц не зависит от y, а вертикальное смещение меняется линейно с глубиной, достигая нуля на дне и максимума на поверхности.

```
DynamicModule[{w = 1, L = 3, k = π/L, a = 0.2, h = 1, X, Y},
  X[u_, v_, t_] = u + 2aCos[wt]Cos[ku];
  Y[u_, v_, t_] = v + 2aCos[wt]Sin[ku]k(v + h);
  Animate[
    ParametricPlot[{{X[u, v, t], Y[u, v, t]}}, {u, -L/2, L/2}, {v, -h, 0},
                   Mesh → 26, PlotRange → {{-L/2, L/2}, {-h, 0.5}},
                   PlotLabel → t, Axes → {True, False}],
  {t, 0, 2π}, AnimationRunning → False]]
```



Заметим, что в нашем примере $\lambda = 2\pi / k = 2\pi /(\pi / L) = 2L = 6$, h=1 и соотношение $h << \lambda$ для волн на неглубокой воде выполняется (приблизительно).

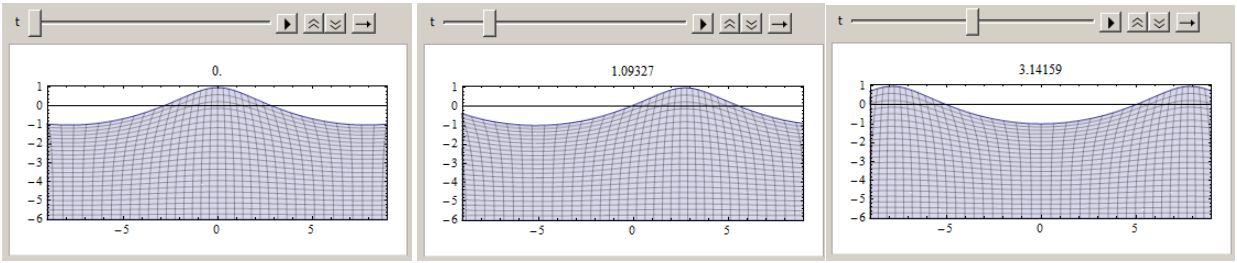
Пример 4.5. Бегущие волны в водоеме конечной глубины, неограниченном в горизонтальном направлении, можно смоделировать уравнениями [1]

$$U = a\sin(wt - k\,x)\left(e^{k\,y} + e^{-2k\,h}e^{-k\,y}\right)$$
$$V = a\cos(wt - k\,x)\left(e^{k\,y} - e^{-2k\,h}e^{-k\,y}\right) \qquad (4)$$

где U и V мгновенные значения смещений частицы жидкости с равновесными координатами x, y в направлениях осей X и Y. Следующий код моделирует движение жидкости в соответствии с уравнениями (4).

```
DynamicModule[{w = 1, L = 6, a = 1, h = 6, k = 0.4, X, Y},
  X[u_, v_, t_] = u + a Sin[wt - ku](Exp[kv] + Exp[-2kh]Exp[-kv]);
  Y[u_, v_, t_] = v + a Cos[wt - ku](Exp[kv] - Exp[-2kh]Exp[-kv]);
  Animate[
    ParametricPlot[{{X[u, v, t], Y[u, v, t]}}, {u, -2L, 2L}, {v, -h, 0},
          Mesh → {49, 21}, PlotRange → {{-1.5L, 1.5L}, {-h, 1.1}},
          PlotLabel → t, Axes → {True, False}],
  {t, 0, 2π}, AnimationRunning → False]]
```

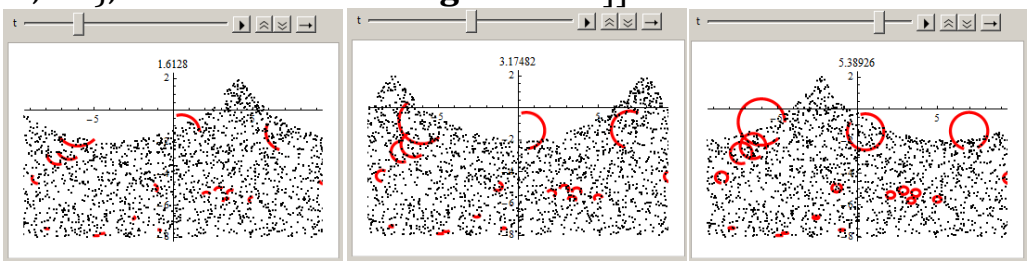
На следующем рисунке представлена двумерная область, имитирующая бегущую волну в последовательные моменты времени.



Из формул (4) следует, что частицы жидкости двигаются по эллипсам $\frac{(X-x)^2}{\left(e^{ky}+e^{-2kh}e^{-ky}\right)^2}+\frac{(Y-y)^2}{\left(e^{ky}-e^{-2kh}e^{-ky}\right)^2}=1$, которые с увеличением глубины постепенно «сплющиваются», а на дне вырождаются в горизонтальные отрезки. Этот факт можно продемонстрировать графически, если среду смоделировать в виде множества точек и нарисовать траектории некоторых из них.

```
DynamicModule[{n = 2000, L = 6, h = 8, a = 2, w = 1, np = 20, k = 0.4,
                ul, vl, X, Y, nl, up, vp, Xp, Yp, Xl, Yl, pt, pp, pl},
  ul = RandomReal[{-1.8L, 1.8L}, n];
  vl = RandomReal[{-h, 0}, n];
  X[u_, v_, t_] = u + aSin[wt - ku](Exp[kv] + Exp[-2kh]Exp[-kv]);
  Y[u_, v_, t_] = v + aCos[wt - ku](Exp[kv] - Exp[-2kh]Exp[-kv]);
  SeedRandom[Floor[SessionTime[ ]]];
  nl = RandomInteger[{1, n}, np];
  up = ul[[nl]]; vp = vl[[nl]];
  Xp[τ_] = X[up, vp, τ];
  Yp[τ_] = Y[up, vp, τ];
  Animate[
    Xl = X[ul, vl, t];
    Yl = Y[ul, vl, t];
    pt = Transpose[{Xl, Yl}];
    pp = ParametricPlot[Transpose[{Xp[τ], Yp[τ]}], {τ, 0, t},
                  PlotStyle → {Red, Thickness[0.01]}];
```

```
pl = ListPlot[pt, PlotStyle → {Black, PointSize[0.005]},
                PlotRange → {{-1.5L, 1.5L}, {-h, 0}}];
Show[pp, pl, PlotRange → {{-1.5L, 1.5L}, {-h, 2.1}},
                AxesOrigin → Automatic, PlotLabel → t],
{t, 0.01, 2π}, AnimationRunning → False]]
```



Если рассмотреть водоем бесконечной глубины, то траектории точек будут окружностями, радиус которых зависит от глубины y. Смоделируйте самостоятельно такое движение.

2.4.5 Одномерные волновые колебания

Поперечные колебания струны, продольные колебания упругого стержня и многие другие явления описываются одномерным волновым уравнением

$$\frac{\partial^2 u(x,t)}{\partial t^2} = a^2 \frac{\partial^2 u(x,t)}{\partial x^2} \qquad (1)$$

Для однозначного определения решения необходимо задать начальные условия

$$u(x,0) = \varphi(x),\ u'_t(x,t) = \psi(x) \qquad (2)$$

и граничные условия, если у колеблющегося объекта имеются границы. Обычно функция $u(x,t)$ представляет смещение из положения равновесия точки x одномерной среды в момент времени t.

Известно общее решение Даламбера задачи (1), (2) для неограниченной прямой [2]

$$u(x,t) = \frac{\varphi(x+at)+\varphi(x-at)}{2} + \frac{1}{2a}\int_{x-at}^{x+at} \psi(\alpha)d\alpha \qquad (-\infty < x < \infty, t \ge 0). \qquad (3)$$

Формула (3) при любых функциях $\varphi(x)$ и $\psi(x)$ дает решение уравнения (1). В данном параграфе во всех задачах начальная скорость колебаний равна нулю $\psi(x) = 0$ и поэтому решение будет иметь вид

$$u(x,t) = \frac{\Phi(x+at)+\Phi(x-at)}{2}, \qquad (4)$$

а функция Ф(x) будет подбираться так, чтобы удовлетворить начальному условию $u(x,0) = \varphi(x)$ и дополнительным условиям на границе. Во всех задачах также будем считать, что параметр a=1.

Пример 5.1. Колебание конечной струны с синусоидальным начальным профилем и нулевой начальной скоростью.

Пусть струна имеет конечную длину L=1 и ось X направлена вправо вдоль струны Если начальные условия имеют вид $u(x,0) = \sin \pi x$, $u'_t(x,0) = 0$ и струна закреплена на концах, т.е. $u(0,t) = u(L,t) = 0$, то решение уравнения (1),

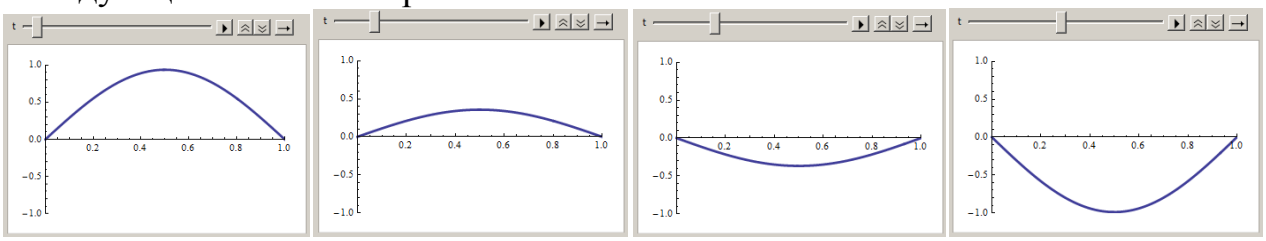
удовлетворяющее указанным условиям, может быть получено из (4) при $\Phi(x)=\sin \pi x$

$$u(x,t)=\frac{\sin(\pi(x+t))+\sin(\pi(x-t))}{2}$$

В следующем коде мы моделируем движение струны в соответствии с этим решением.

```
DynamicModule[{u, f},
  f[x_] = Sin[πx];
  u[x_, t_] = (f[x + t] + f[x - t])/2;
  Animate[Plot[Evaluate[u[x, t]], {x, 0, 1},
            PlotStyle → Thickness[0.01], PlotPoints → 1000,
            PlotRange → {{0, 1}, {-1, 1}}],
   {t, 0, 2}, AnimationRunning → False, DisplayAllSteps → True]]
```

На следующем рисунке показана панель анимации в начальный и несколько последующих моментов времени.



Пример 5.2. *Колебание полубесконечной струны, закрепленной на конце.*

Рассмотрим задачу о распространении начального возмущения $u(x,0)=\varphi(x)$ по полубесконечной струне с закрепленным концом *x=0.* Начальная скорость равна нулю $\psi(t)=0$ и граничное условие имеет вид $u(0,t)=0$. Можно показать, что, если начальную функцию $\varphi(x)$ продолжить с луча x≥0 нечетно на всю вещественную ось и подставить в (4), то получим решение поставленной задачи [2]. Это значит, что полученная таким образом функция $u(x,t)$, рассматриваемая только для x≥0, будет удовлетворять уравнению, граничному и обоим начальным условиям.

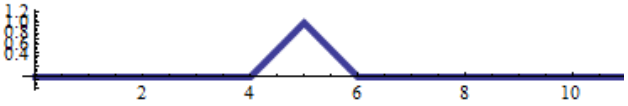
Нечетное продолжение $\Phi(x)$ любой функции $\varphi(x)$, заданной только для положительных *x*, на всю вещественную ось можно получить по формуле $\Phi(x)=sign(x)\,\varphi(|x|)$. Тогда из (4) получаем

$$u(x,t)=\frac{\varphi(x+at)+sign(x-at)\varphi(|x-at|)}{2} \qquad (5)$$

Пусть начальное отклонение струны равно нулю везде, кроме интервала (4,6), в котором функция φ(x) изображается равнобедренным треугольником единичной высоты. На следующем рисунке показана форма начального профиля струны.

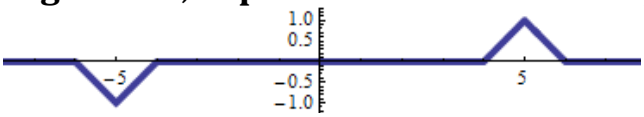
```
φ[x_] = 1/2 Abs[x - 4] - Abs[x - 5] + 1/2 Abs[x - 6];
```

```
Plot[φ[x], {x, 0, 12}, PlotStyle → Thickness[0.01],
    PlotPoints → 1000, PlotRange → All, AspectRatio → Automatic]
```



Ниже показан график функции $\Phi(x)$, являющейся нечетным продолжением функции $\varphi(x)$.

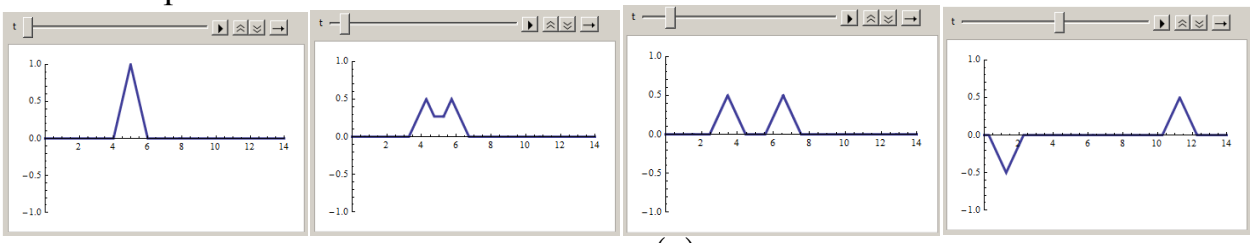
```
f[x_] = Sign[x]φ[Abs[x]];
Plot[f[x], {x, −8, 8}, PlotStyle → Thickness[0.01], PlotPoints → 1000,
    PlotRange → All, AspectRatio → Automatic]
```



Следующий код моделирует движение струны, используя формулу (5).

```
DynamicModule[{u},
  u[x_, t_] = (f[x + t] + f[x − t])/2;
  Animate[
    Plot[Evaluate[u[x, t]], {x, 0, 14},
        PlotStyle → Thickness[0.01], PlotPoints → 1000,
        PlotRange → {{0, 14}, {−1, 1}}],
    {t, 0, 12}, AnimationRunning → False, DisplayAllSteps → True]]
```

Ниже показана панель анимации в начальный и несколько последующих моментов времени.



Заметим, что если начальная функция $\varphi(x)$ не дифференцируема в некоторых точках, то полученное решение тоже не дифференцируемо в некоторых точках. Такое решение называют обобщенным.

Пример 5.3. *Колебание полубесконечной струны со свободным левым концом*.

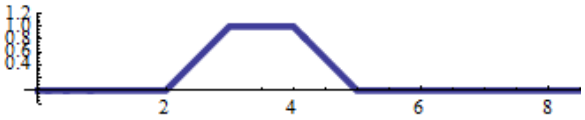
Рассмотрим задачу о распространении начального возмущения $u(x,0)=\varphi(x)$ по полубесконечной струне со свободным левым концом $x=0$. Начальная скорость равна нулю $\psi(t)=0$ и граничное условие имеет вид $u'_x(0,t)=0$. Можно показать, что, если начальную функцию $\varphi(x)$ продолжить с луча $x\geq 0$ четно на всю вещественную ось и подставить в (4), то получим решение поставленной задачи [2]. Функцию $u(x,t)$ следует рассматриваемая только для $x\geq 0$ и она будет удовлетворять уравнению (1), граничному и обоим начальным условиям.

Четное продолжение $\Phi(x)$ на вещественную ось любой функции $\varphi(x)$, заданной только для $x \geq 0$, можно получить по формуле $\Phi(x) = \varphi(|x|)$. Тогда из (4) получим

$$u(x,t) = \frac{\varphi(x+at) + \varphi(|x-at|)}{2} \qquad (6)$$

Пусть начальное отклонение струны равно нулю везде, кроме интервала (2,5) в котором функция φ(x) изображается равнобедренной трапецией высотой 1. На следующем рисунке показан график начального профиля струны.

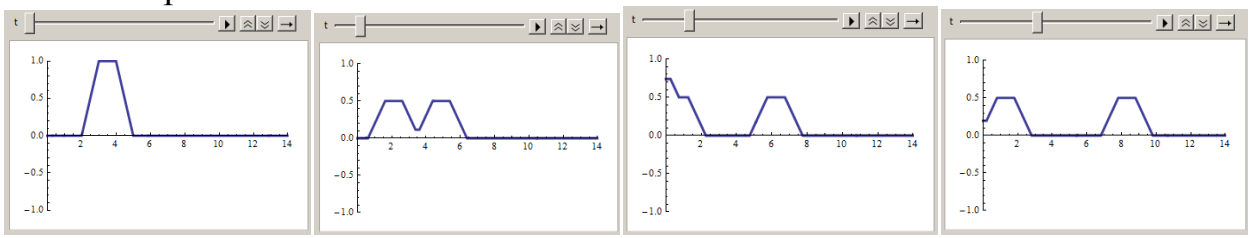
```
φ[x_] = 1/2 Abs[x - 2] - 1/2 Abs[x - 3] - 1/2 Abs[x - 4] + 1/2 Abs[x - 5];
Plot[φ[x], {x, 0, 10}, PlotStyle → Thickness[0.01], PlotPoints → 1000,
                    PlotRange → All, AspectRatio → Automatic]
```



Следующий код моделирует движение струны, используя формулу (6).

```
DynamicModule[{u, φ, f},
  φ[x_] = 1/2 Abs[x - 2] - 1/2 Abs[x - 3] - 1/2 Abs[x - 4] + 1/2 Abs[x - 5];
  f[x_] = φ[Abs[x]];
  u[x_, t_] = (f[x + t] + f[x - t])/2;
  Animate[
    Plot[Evaluate[u[x, t]], {x, 0, 14}, PlotStyle → Thickness[0.01],
                    PlotPoints → 1000, PlotRange → {{0, 14}, {-1, 1}}],
  {t, 0, 12}, AnimationRunning → False, DisplayAllSteps → True]]
```

Ниже показана панель анимации в начальный и несколько последующих моментов времени.



Пример 5.4. *Колебание конечной струны с закрепленными концами*.

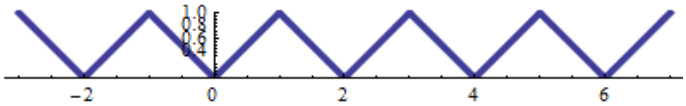
Рассмотрим задачу о колебании конечной струны длины L с закрепленными концами, начальным смещением $u(x,0) = \varphi(x)$ и нулевой начальной скорость $\psi(t) = 0$. Граничные условия имеют вид $u(0,t) = 0$, $u(L,t) = 0$. Если начальную функцию $\varphi(x)$, обращающуюся в ноль на концах отрезка [0, L], продолжить на всю вещественную ось нечетно и периодически (с отрезка [0, L] на отрезок [-L, 0] нечетно, а затем с отрезка [-L, L] периодически на всю ось) и, созданную функцию $\Phi(x)$ подставить в (4), то получим решение рассматриваемой задачи [3]. Функция $u(x,t)$, рассматриваемая только для $0 \leq x \leq L$, будет удовлетворять обоим начальным и обоим граничным условиям.

Нечетное периодическое продолжение $\Phi(x)$ любой функции $\varphi(x)$, заданной на отрезке $0 \le x \le L$, на всю вещественную ось можно получить по формуле [3]

$$\Phi(x) = (-1)^{\left[\frac{x}{L}\right]} \varphi(stc(x,2L)), \tag{6}$$

где [x] (квадратные скобки) представляет функцию взятия наибольшего целого, не превосходящего x, а $stc(x,w)$ является периодической пилообразной непрерывной кусочно-линейной функцией, определяемой формулой $stx(x,w) = \left| x - w\left[\frac{x}{w} + \frac{1}{2}\right]\right|$ или $stc(x,w) = \frac{w}{2\pi}\arccos\left(\cos\frac{2\pi x}{w}\right)$, где w является периодом этой функции.

```
Stc[x_, w_] = Abs[x - w Floor[x/w + 1/2]];
Plot[Stc[x, 2], {x, -3, 7}, , AspectRatio → Automatic]
```



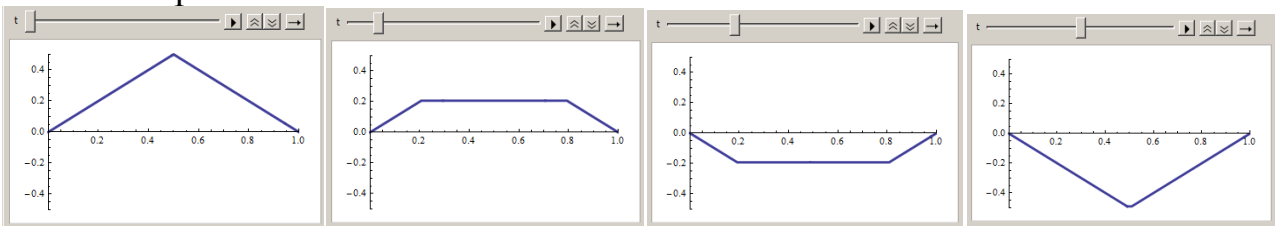
Из (4) имеем

$$u(x,t) = \frac{(-1)^{\left[\frac{x+at}{L}\right]}\varphi(stc(x+at,2L)) + (-1)^{\left[\frac{x-at}{L}\right]}\varphi(stc(x-at,2L))}{2} \tag{7}$$

Пусть L=1 и начальная функция φ(x) имеет треугольную форму, как показано на следующем рисунке слева. Следующий код моделирует движение точек струны, смещение $u(x,t)$ которых задается формулой (7).

```
DynamicModule[{L = 1, φ, F, u},
  φ[x_] = 1/2 - Abs[x - 1/2];
  F[x_] = (-1)^Floor[x/L] φ[Stc[x, 2L]];
  u[x_, t_] = (F[x + t] + F[x - t])/2;
  Animate[
    Plot[Evaluate[u[x, t]], {x, 0, L}, PlotStyle → Thickness[0.01],
         PlotPoints → 1000, PlotRange → {{0, L}, {-0.5, 0.5}}],
  {t, 0, 2}, AnimationRunning → False, DisplayAllSteps → True]]
```

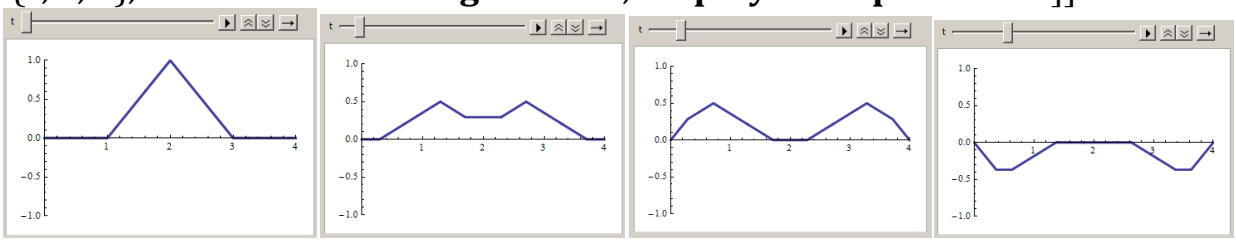
Ниже показана панель анимации в начальный и несколько последующих моментов времени.



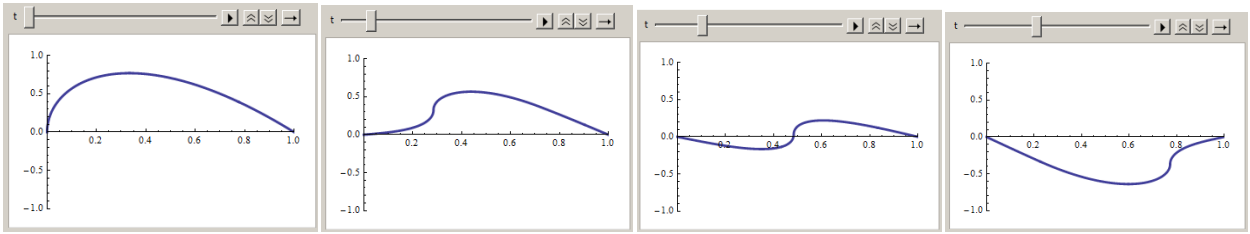
Формула (7) дает решение задачи о колебании конечной струны длины L, закрепленной на концах, имеющей нулевую начальную скорость и начальное смещение $u(x,0) = \varphi(x)$. Приведем несколько примеров колебаний при

различных функциях $\varphi(x)$. Во всех примерах левые панели анимации показаны для момента времени t=0, т.е. на них показаны начальные профили струн.

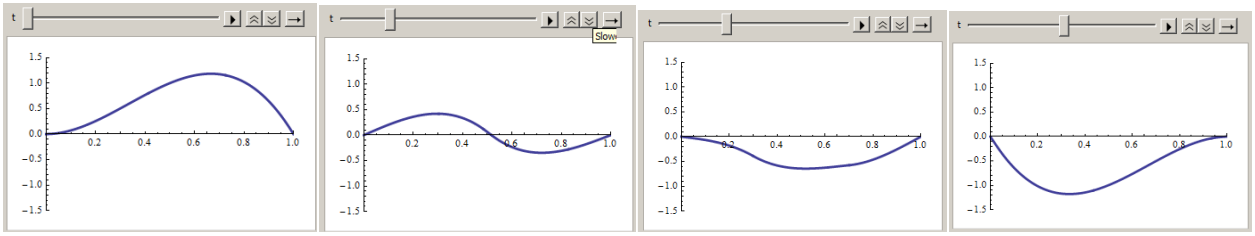
```
DynamicModule[{L = 4, f, F, u},
  f[x_] = 1/2 Abs[x - 1] - Abs[x - 2] + 1/2 Abs[x - 3];
  F[x_] = (-1)^Floor[x/L] f[Stc[x, 2L]];
  u[x_, t_] = (F[x + t] + F[x - t])/2;
  Animate[
    Plot[Evaluate[u[x, t]], {x, 0, L}, PlotStyle → Thickness[0.01],
                PlotPoints → 1000, PlotRange → {{0, L}, {-1, 1}}],
  {t, 0, 8}, AnimationRunning → False, DisplayAllSteps → True]]
```



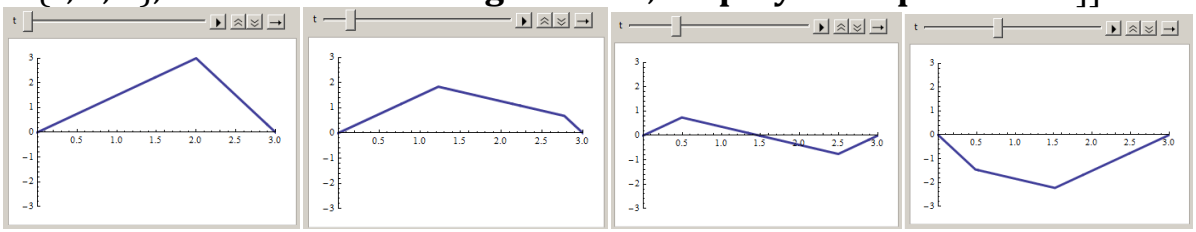
```
DynamicModule[{L = 1, f, F, u},
  f[x_] = 2√x (1 - x);
  F[x_] = (-1)^Floor[x/L] f[Stc[x, 2L]];
  u[x_, t_] = (F[x + t] + F[x - t])/2;
  Animate[
    Plot[Evaluate[u[x, t]], {x, 0, L}, PlotStyle → Thickness[0.01],
                PlotPoints → 1000, PlotRange → {{0, L}, {-1, 1}}],
  {t, 0, 2}, AnimationRunning → False, DisplayAllSteps → True]]
```



```
DynamicModule[{L = 1, f, F, u},
  f[x_] = 8x^2 (1 - x);
  F[x_] = (-1)^Floor[x/L] f[Stc[x, 2L]];
  u[x_, t_] = (F[x + t] + F[x - t])/2;
  Animate[
    Plot[Evaluate[u[x, t]], {x, 0, L}, PlotStyle → Thickness[0.01],
                PlotPoints → 1000, PlotRange → {{0, L}, {-1.5, 1.5}}],
  {t, 0, 2}, AnimationRunning → False, DisplayAllSteps → True]]
```



```
DynamicModule[{L = 3, f, F, u},
 f[x_] = -3/4 x + 9/2 - 9/4 Abs[x - 2];
 F[x_] = (-1)^Floor[x/L] f[Stc[x, 2L]];
 u[x_, t_] = (F[x + t] + F[x - t])/2;
 Animate[
   Plot[Evaluate[u[x, t]], {x, 0, L}, PlotStyle -> Thickness[0.01],
                PlotPoints -> 1000, PlotRange -> {{0, L}, {-3, 3}}],
 {t, 0, 6}, AnimationRunning -> False, DisplayAllSteps -> True]]
```



Заметим, что во всех приведенных выше примерах используется определение функции $stc(x,w)$, приведенное нами ранее.

Пример 5.5. Решим задачу о колебание конечной струны с закрепленным левым концом и свободным правым. Для этого надо решить уравнение колебаний (1) с начальными условиями (2) и граничными условиями

$$u(0,t)=0,\ u'_x(L,t)=0 \qquad (8)$$

Известно, если начальные данные в задаче о распространении колебаний на неограниченной прямой являются нечетными функциями относительно некоторой точки x_0, то соответствующее решение в этой точке равно нулю.

Если начальные данные в задаче о распространении колебаний на неограниченной прямой являются четными функциями относительно некоторой точки x_0, то производная по x соответствующего решения в этой точке равна нулю.

Рассмотрим функцию $\varphi(x)$ заданную на отрезке [*0,L*] такую, что $\varphi(0)=0$. Продолжим ее четно относительно точки $x=L$ на отрезок [*L,2L*], т.е. построим функцию $\varphi_1(x)$, $x\in[0,2L]$ такую, что для $x\in[0,L]$, $\varphi_1(x)=\varphi(x)$, а для $x\in[L,2L]$ $\varphi_1(x)=\varphi(2L-x)$. Функцию $\varphi_1(x)$ с отрезка [0,2*L*] продолжим нечетно на отрезок $[-2L,0]$, т.е. построим функцию $\varphi_2(x)$, $x\in[-2L,2L]$ такую, что для $x\in[0,2L]$, $\varphi_2(x)=\varphi_1(x)$, а для $x\in[-2L,0]$, $\varphi_2(x)=-\varphi_1(-x)$. Полученную функцию $\varphi_2(x)$ с отрезка [*-2L,2L*] продолжим периодически на всю ось с периодом *4L*. В результате получим периодическую функцию $\Phi(x)$ которая на отрезке $x\in[0,L]$ совпадает с $\varphi(x)$, является четной относительно точки $x=L$ и

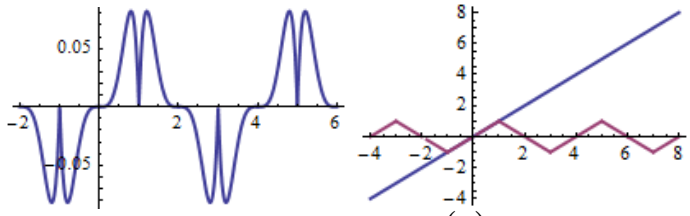
нечетной относительно точки x=0. Назовем описанное продолжение функции $\varphi(x)$ четно-нечетным периодическим продолжением. В силу сказанного выше, подстановка такой функции Φ(x) в формулу (4) даст решение поставленной задачи (1), (2), (8).

Известно [4], что для любой функции $\varphi(x)$ заданной на отрезке [0,L] ($L>0$), такой что $\varphi(0)=0$, формула четно-нечетного периодического продолжения имеет следующий вид

$$\Phi(x)=(-1)^{\left[\frac{x}{2L}\right]}\varphi(stc(x,2L)) \qquad (9)$$

Вот пример четно-нечетного продолжения функции $f(x)=x^4(1-x)$ с отрезка [0,1] на всю вещественную ось (следующий рисунок слева)

```
L = 1; f[x_] = x^4 (L - x);
F[x_] = (-1)^Floor[x/(2L)] f[Stc[x, 2L]];
Plot[F[x], {x, -2, 6}, PlotRange → All, PlotStyle → Thickness[0.01]]
```



Четно – нечетное продолжение функции $f(x)=x$ с отрезка [0,1] на всю вещественную ось показано на предыдущем рисунке справа, где кроме продолженной функции, для сравнения мы изобразили и исходную.

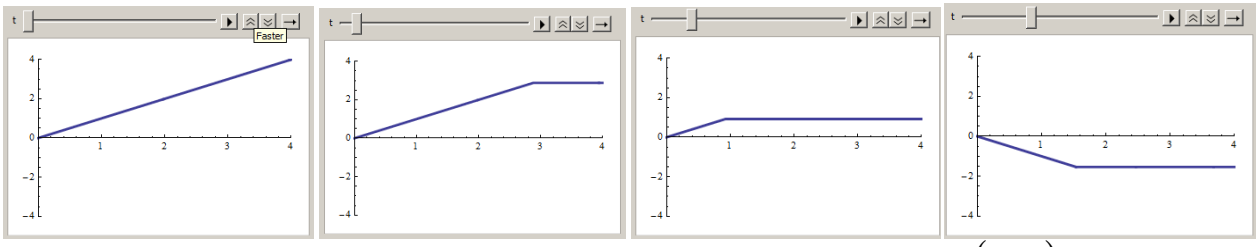
Таким образом , рением задачи (1), (2), (8) при $\psi(x)=0$ будет функция [4]

$$u(x,t)=\frac{(-1)^{\left[\frac{x+at}{2L}\right]}\varphi(stc(x+at,2L))+(-1)^{\left[\frac{x-at}{2L}\right]}\varphi(stc(x-at,2L))}{2} \qquad (10)$$

Пусть L=4 и начальная функция $\varphi(x)=x$. На следующем рисунке слева показана начальная форма струны. Код, приведенный ниже, моделирует движение точек струны, смещение которых $u(x,t)$ задается формулой (10).

```
DynamicModule[{L = 4, f, F, u},
  f[x_] = x;
  F[x_] = (-1)^Floor[x/(2L)] f[Stc[x, 2L]];
  u[x_, t_] = (F[x + t] + F[x - t])/2;
  Animate[
    Plot[Evaluate[u[x, t]], {x, 0, L}, PlotStyle → Thickness[0.01],
                 PlotPoints → 1000, PlotRange → {{0, L}, {-L, L}}],
  {t, 0, 16}, AnimationRunning → False, DisplayAllSteps → True]]
```

На следующих рисунках показана панель анимации в начальный и несколько последующих моментов времени.



Заметим, что в коде используется определение функции $stc(x,w)$, приведенное нами ранее.

Пример 5.6. Продольные колебания стержня постоянного поперечного сечения описываются уравнением (1) с начальными условиями (2) и граничными условиями, зависящими от способа закрепления концов [5].

Пусть круглый стержень длиной L, закрепленный на одном конце и свободный на другом, подвержен линейному растяжению. Начало координат расположим в центре левого (фиксированного) сечения, и ось X направим вправо вдоль оси стержня. Начальное смещение будет иметь вид $u(x,0)=\varphi(x)=k\,x$, где k некоторая постоянная. В начальный момент времени растянутый конец освобождается и стержень начинает сжиматься, а затем удлиняться – начинаются продольные колебания. Такие колебания будут описываться уравнением (1), где $u(x,t)$ является продольным смещением в момент времени t сечения стержня, имеющего в нулевой момент времени координату x. Начальные условия будут иметь вид

$$u(x,0)=k\,x,\ u'_t(x,0)=0\ , \qquad (11)$$

а граничные условия будут иметь вид (8). Решение этой задачи нами построено в предыдущем примере. Здесь мы смоделируем движение круглого стержня, сечения которого смещаются в соответствии с решением $U(x,t)$ задачи (1), (8), (11). Для функции решения мы используем обозначение U, чтобы не путать его с параметром u в уравнениях поверхности.

Пусть параметрические уравнения поверхности стержня в состоянии покоя (нулевой момент времени) имеют вид $x=x(u,v),\ y=y(u,v),\ z=z(u,v)$. Все точки одного и того же поперечного сечения стержня смещаются в момент t на одну и ту же величину $U(x,t)$. Поэтому в момент t они будут иметь продольную координату $X=x(u,v)+U(x(u,v),t)$. В плоскости сечения смещения точек нет, поэтому y и z координаты точек сечения не меняются. В результате в момент времени t точки стержня, имевшие при t=0 координаты $x=x(u,v),\ y=y(u,v),\ z=z(u,v)$, будут иметь координаты

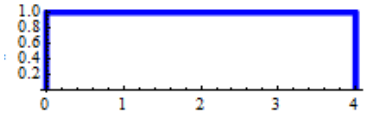
$$X=x(u,v)+U(x(u,v),t),\ Y=y(u,v),\ Z=z(u,v). \qquad (12)$$

В следующем коде мы моделируем продольные колебания круглого стержня радиуса r, точки поверхности которого в момент t имеют координаты (12).

Круглый стержень в п.2.2.3 мы строили, как поверхность вращения ломаной, имеющей форму «скобы». Параметрическое уравнение «скобы» имеет вид

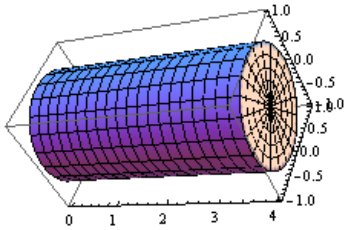
$\boldsymbol{L=4;\ r=1;}$

```
xc[t_] = L/2 + 1/2 Abs[t - r] - 1/2 Abs[t - r - L];
zc[t_] = (r + L/2) - 1/2 Abs[t - r] - 1/2 Abs[t - r - L];
ParametricPlot[{xc[t], zc[t]}, {t, 0, L + 2r},
               PlotStyle → {Thickness[0.02], Blue}]
```



Параметрическое уравнение поверхности вращения кривой $x_c(u), y_c(u)$ вокруг оси X имеет вид $x(u,v)=x_c(u),\ y(u,v)=z_c(u)\cos v,\ z(u,v)=z_c(u,v)\sin v$.

```
x[u_, v_] = xc[u];
y[u_, v_] = Cos[v] zc[u];
z[u_, v_] = Sin[v] zc[u];
ParametricPlot3D[{x[u, v], y[u, v], z[u, v]}, {u, 0, L + 2r}, {v, 0, 2π},
                 Mesh → Full, PlotPoints → {31, 21}, BoxRatios → {L, 2r, 2r}]
```

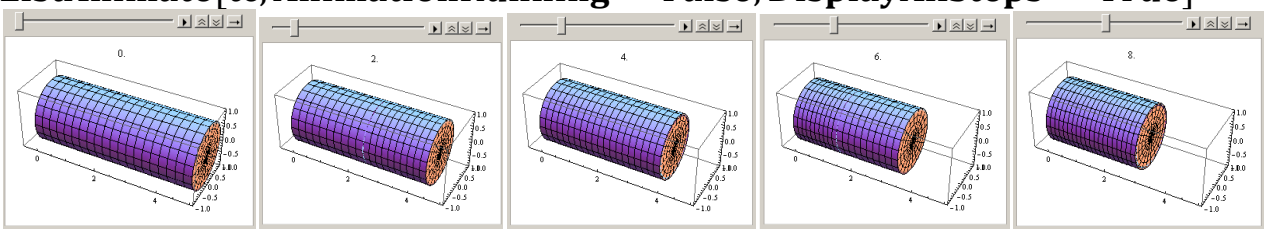


Записываем определение координатных функций (12).

```
Stc[x_, w_] = Abs[x - w Floor[x/w + 1/2]];
f[x_] = x/L;
F[x_] = (-1)^Floor[x/(2L)] f[Stc[x, 2L]];
U[x_, t_] = (F[x + t] + F[x - t])/2;
X[u_, v_, t_] = x[u, v] + U[x[u, v], t];
Y[u_, v_, t_] = y[u, v];
Z[u_, v_, t_] = z[u, v];
```

Создаем список **tc** изображений стержня в различные моменты времени и анимируем его с помощью функции `ListAnimate`.

```
tc = Table[ParametricPlot3D[
                {X[u, v, t], Y[u, v, t], Z[u, v, t]}, {u, 0, L + 2r}, {v, 0, 2π},
                Mesh → Full, PlotPoints → {31, 21},
                PlotRange → {{-0.5, L + 1.1}, {-r, r}, {-r, r}}, PlotLabel → t],
           {t, 0, 4L - 0.25, 0.25}];
ListAnimate[tc, AnimationRunning → False, DisplayAllSteps → True]
```



Обратите внимание на то, что стержень не просто сжимается/разжимается, но в отдельных частях сжимается/разжимается координатная сетка на его поверхности.

Причина по которой мы использовали функцию `ListAnimate`, а не `Animate`, состоит в том, что построение анимации большого количества 3D графиков занимает у системы много времени. В примере процесс разделен на два шага. Первый состоит в создании списка изображений. Второй шаг состоит воспроизведении этого вписка, что происходит более гладко.

Отметим, что приведенный код не собран нами в динамический модуль для того, чтобы продемонстрировать результаты работы отдельных блоков кода. Однако, вы можете сделать это самостоятельно.

Пример 5.7. Продольные колебания пружины. Уравнение продольных колебаний для стержня и пружины записываются одинаково [2].

Пусть спиральная пружина длиной L, закрепленная на одном конце и свободная на другом, подвержена линейному растяжению. Начало координат расположим в на левом краю пружины, и ось X направлена вправо вдоль ее оси. Если к свободному концу пружины приложить растягивающее усилие, то деформация будет линейной. Допустим, что усилие таково, что правый конец сместится на единицу. Тогда начальное смещение будет иметь вид $u(x,0)=x/L$. В начальный момент времени растянутый конец освобождается и пружина начинает сжиматься и разжиматься – начинаются продольные колебания. Такие колебания описываются уравнением (1), где $u(x,t)$ является продольным смещением в момент времени t точек пружины, имевших в нулевой момент времени координату x.

Решение уравнения (1) с начальными условиями (11) и граничными условиями (8) нами уже построено ранее. Здесь мы смоделируем движение спиральной пружины, точки которой смещаются в соответствии с решением $U(x,t)$ задачи (1), (8), (11).

Пусть параметрические уравнения спиральной кривой, представляющей пружину, имеют вид $x=x(u),\ y=y(u), z=z(u)$. Точка пружины в момент времени t смещается в продольном направлении на величину $U(x,t)$. Поэтому в момент t она будет иметь новую координату $X=x(u)+U(x(u),t)$. Координаты y и z точки не меняются. В результате в момент времени t точка пружины, имевшая при t=0 координаты $x=x(u),\ y=y(u), z=z(u)$, будет иметь координаты

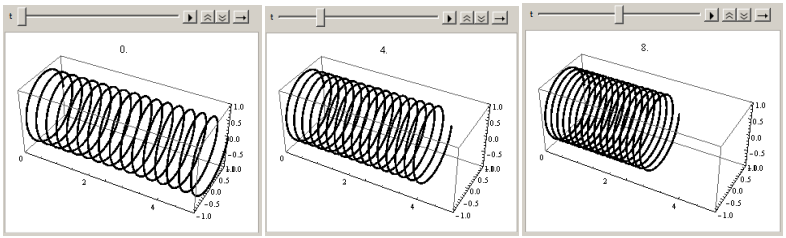
$$X=x(u)+U(x(u),t),\ Y=y(u),\ Z=z(u). \qquad (13)$$

В следующем коде мы моделируем продольные колебания пружины радиуса r, точки которой в момент t имеют координаты (13). Пружину рисуем в виде пространственной круговой спиральной кривой длиной L=4 с n=4 витками на единице длины.

```
DynamicModule[{L = 4, n = 4, x, y, z, f, F, U, X, Y, Z},
  x[w_] = w;
  y[w_] = Cos[2 * Pi * w * n];
```

```
z[w_] = Sin[2 * Pi * w * n];
f[x_] = z/L;
F[x_] = (-1)^Floor[x/(2L)] f[Stc[x, 2L]];
U[x_, t_] = (F[x - t] + F[x + t])/2;
X[w_, t_] = x[w] + U[x[w], t];
Y[w_, t_] = y[w];
Z[w_, t_] = z[w];
Animate[
   ParametricPlot3D[Evaluate[{X[w, t], Y[w, t], Z[w, t]}], {w, 0, L},
       PlotStyle → Thickness[0.01], AspectRatio → Automatic,
       PlotPoints → 1000, PlotRange → {{0, L + 1}, {-1, 1}, {-1, 1}},
       PlotLabel → t],
  {t, 0, 4L}, AnimationRunning → False, DisplayAllSteps → True]]
```

Форма пружины в моменты времени $t = 0, 4, 8$ показана на следующем рисунке.



Пример 5.8. *Крутильные колебания кругового стержня*.

Рассмотрим круглый вертикальный стержень длиной L и радиуса r, зафиксированный в нижнем сечении и закрученный парой сил в верхнем. Круговые сечения стержня остаются круговыми в процессе закручивания, а диаметр сечений и расстояния между ними не меняется при условии, что угол закручивания мал. Начало координат расположим в центре основания стержня и ось Z направим вдоль его оси вертикально вверх. В начальный момент верхнее сечение освобождается и в стержне возникают крутильные колебания. Известно [5], что для поперечного сечения с координатой z угол поворота $\theta(z,t)$удовлетворяет волновому уравнению (1).

Начальный угол поворота $\theta(z,0)$ будет линейно меняться вдоль стержня $\theta(z,0) = k\,z$, где постоянная k зависит от свойств материала стержня и пары закручивающих сил. Пусть в параметрических уравнениях поверхности цилиндра $x(u,v), y(u,v), z(u,v)$ параметр v представляет угол. Тогда в уравнениях поверхности закрученного цилиндра этот параметр должен быть увеличен на величину угла θ. Угол поворота сечений является функцией координаты z и момента времени t, т.е. $\theta = U(z,t)$. В результате уравнения поверхности закрученного стержня будут иметь вид

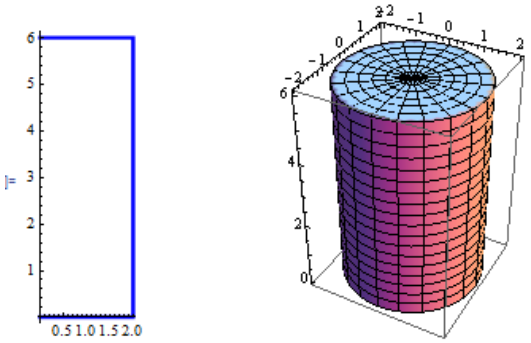
$$X = x(u, v + U(z(u,v),t)),\ Y = y(u, v + U(z(u,v),t)),\ Z = z(u, v + U(z(u,v),t)), \quad (14)$$

а функцию $U(z,t)$ следует искать как решения волнового уравнения (1) с начальными условиями $U(z,0) = k\,z, U'_t(z,0) = 0$ и граничными условиями

$U(0,t)=0$ (нижнее сечение закреплено, поворота нет), $U'_x(L,t)=0$ (верхнее сечение свободно) [5]. Решение этой задачи мы уже построили в примере 5.5. Используем его для графического представления крутильных колебаний.

Вначале составляем параметрические уравнения поверхности цилиндрического стержня, как поверхности вращения ломаной в форме «скобы» вокруг оси Z.

```
L = 6; r = 2;
xc[t_] = (r + L/2) - 1/2 Abs[t - r] - 1/2 Abs[t - r - L];
zc[t_] = L/2 + 1/2 Abs[t - r] - 1/2 Abs[t - r - L];
ParametricPlot[{xc[t], zc[t]}, {t, 0, L + 2r},
        PlotStyle → {Thickness[0.02], Blue}]  (* рисунок скобы слева *)
x[u_, v_] = Cos[v]xc[u];
y[u_, v_] = Sin[v]xc[u];
z[u_, v_] = zc[u];
ParametricPlot3D[Evaluate[{x[u, v], y[u, v], z[u, v]}],
        {u, 0, L + 2r}, {v, 0, 2π}, Mesh → Full, PlotPoints → {31, 21},
        BoxRatios → {2r, 2r, L}]    (* рисунок цилиндра справа *)
```



Затем строим решение уравнения крутильных колебаний (1) с начальными $U(z,0)=z/L, U'_t(z,0)=0$ и граничными $U(0,t)=0$, $U'_x(L,t)=0$ условиями.

```
Stc[x_, w_] = Abs[x - wFloor[x/w + 1/2]];
f[x_] = x/L;
F[x_] = (-1)^Floor[x/(2L)] f[Stc[x, 2L]];
U[x_, t_] = (F[x + t] + F[x - t])/2;
```

Составляем параметрические уравнения поверхности закрученного стержня в соответствии с формулами (14)

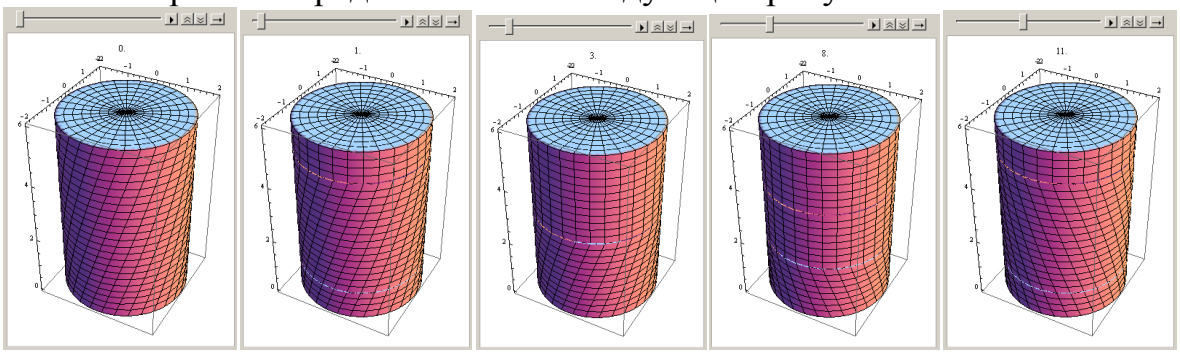
```
X[u_, v_, t_] = x[u, v + U[z[u, v], t]];
Y[u_, v_, t_] = y[u, v + U[z[u, v], t]];
Z[u_, v_, t_] = z[u, v + U[z[u, v], t]];
```

Создаем список **tc** изображений стержня в различные моменты времени и анимируем его с помощью функции `ListAnimate`.

```
tc = Table[ParametricPlot3D[
        Evaluate[{X[u, v, t], Y[u, v, t], Z[u, v, t]}], {u, 0, L + 2r}, {v, 0, 2π},
        PlotPoints → {41, 31}, Mesh → Full,
```

$$\textbf{PlotRange} \to \{\{-r, r\}, \{-r, r\}, \{-0.1, L + 0.1\}\}, \textbf{PlotLabel} \to t],$$
$$\{t, 0, 4L - 0.25, 0.25\}];$$
$$\textbf{ListAnimate}[\text{tc}, \textbf{AnimationRunning} \to \textbf{False}, \textbf{DisplayAllSteps} \to \textbf{True}]$$

Изображения крутящегося цилиндра в начальный и некоторые последующие моменты времени представлены на следующем рисунке.



Обратите внимание, что геометрическое тело – цилиндр остается цилиндром, однако меняется координатная сеть на его поверхности. Она показывает, как закручены те или иные сечения цилиндра.

2.4.6 Задачи сопротивления материалов и теории упругости

Пример 6.1. *Изгиб балки равномерно распределенной нагрузкой*.

Пусть начало координат находится на левом конце балки и ось X направим вдоль оси балки вправо, ось Y направим вертикально вверх. Под действием приложенных к балке вертикальных сил она прогибается.

Уравнение изгибающего момента балки имеет вид

$$\frac{d^2M}{dx^2} = -q, \tag{1}$$

где q внешняя сила, рассчитанная на единицу длины, а $M(x)$ изгибающий момент в балке. Для определения прогиба балки нужно решать уравнение

$$E\,J\,\frac{d^2u(x)}{d\,x^2} = -M(x), \tag{2}$$

где J – момент инерции поперечного сечения балки, E – модуль упругости материала балки, $u(x)$ – отклонение нейтральной оси балки от положения равновесия. Из (1) и (2) получаем дифференциальное уравнение прогиба

$$\frac{d^2}{dx^2}\left(EJ\,\frac{d^2u}{dx^2}\right) = q(x) \tag{3}$$

Для свободно опертой на краях балки граничные условия имеют вид

$$u(0) = 0, u(L) = 0, u''(0) = 0, u''(L) = 0 \tag{4}$$

При ином закреплении граничные условия будут другими.

Рассмотрим балку длины L и высотой/толщиной h, свободно опертую на концах. Известно [6], что прогиб такой балки определяется по формуле

$$u(x) = \frac{q}{24\,E\,J}\left(L^3x - 2\,L\,x^3 + x^4\right), \tag{5}$$

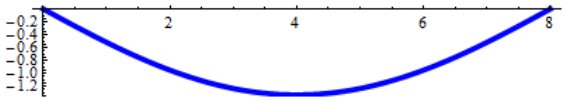
и

$$M(x)=\frac{q\,x}{2}(L-x) \qquad (6)$$

Легко проверить, что функция $u(x)$ удовлетворяет дифференциальному уравнению (3) и граничным условиям (4), а функция $M(x)$ получается из (2) подстановкой в нее функции $u(x)$.

В соответствии с (5) нагруженная балка может быть представлена как изогнутый отрезок.

```
x =.; L = 8; h = 2; Em = 10; J = 4; q = -1;  (* Em – модуль упругости *)
U[x_] = q/(24 * Em * J) * (L^3 * x - 2 * L * x^3 + x^4);
Plot[U[x], {x, 0, L}, PlotStyle → {Thickness[0.01], Blue},
                      AspectRatio → Automatic]
```

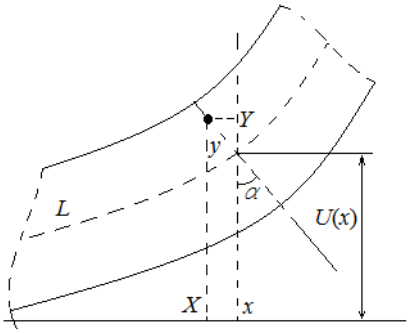


Здесь для модуля упругости мы использовали идентификатор *Em*, поскольку символ *E* в системе защищен (он используется для обозначения основания натуральных логарифмов). Нагрузке q мы присвоили отрицательное значение, поскольку она направлена вниз, а ось Y мы направили вверх.

После выполнения одномерного расчета смоделируем поле напряжений внутри балки. Для этого балку рассмотрим как двумерную область, параметрические уравнения которой имеют вид

$$x(u,v)=u,\ \ y(u,v)=v\ \ (0\le u\le L,-h/2\le v\le h/2).$$

Используя уравнения плоской области $x(u,v), y(u,v)$ недеформированной балки и уравнение $U(x)$ ее нейтральной оси, можно написать параметрическое уравнение области деформированной балки. В соответствии с гипотезой плоских сечений, все поперечные сечения, которые были перпендикулярны нейтральной оси балки до деформирования, остаются перпендикулярными нейтральной оси балки после ее деформирования.



Точки, имевшие до деформирования координаты (x,y), переходят в точки с координатами (X,Y), где $X=x-y\sin\alpha$, $Y=U(x)+y\cos\alpha$ (см. рисунок). Но $tg\,\alpha=U'(x)$ и, поэтому, $X=x-\dfrac{U'_x(x)\,y}{\sqrt{1+(U'_x(x))^2}}$ и $Y=U(x)+\dfrac{y}{\sqrt{1+(U'_x(x))^2}}$. Тогда функции $X(u,v), Y(u,v)$, представляющие параметрические уравнения области изогнутой балки, будут иметь вид

$$X(u,v)=x(u,v)-\frac{U'_x(x(u,v))y(u,v)}{\sqrt{1+(U'_x(x(u,v)))^2}},$$
$$Y(u,v)=U(x(u,v))+\frac{y(u,v)}{\sqrt{1+(U'_x(x(u,v)))^2}} \quad (7)$$

Строим область деформированной балки.

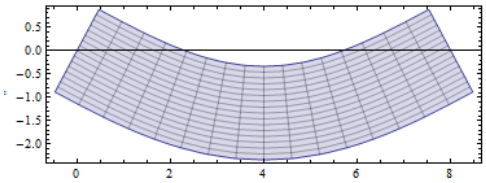
$\boldsymbol{x}[\mathbf{u\_},\mathbf{v\_}]=\boldsymbol{u};\quad \boldsymbol{y}[\mathbf{u\_},\mathbf{v\_}]=\boldsymbol{v};$

$\mathbf{U1}=\boldsymbol{U'};$

$$\boldsymbol{X}[\mathbf{u\_},\mathbf{v\_}]=\boldsymbol{x}[\boldsymbol{u},\boldsymbol{v}]-\frac{\mathbf{U1}[\boldsymbol{x}[\boldsymbol{u},\boldsymbol{v}]]\,\boldsymbol{y}[\boldsymbol{u},\boldsymbol{v}]}{\sqrt{\mathbf{1}+\mathbf{U1}[\boldsymbol{x}[\boldsymbol{u},\boldsymbol{v}]]^{\mathbf{2}}}};$$

$$\boldsymbol{Y}[\mathbf{u\_},\mathbf{v\_}]=\boldsymbol{U}[\boldsymbol{x}[\boldsymbol{u},\boldsymbol{v}]]+\frac{\boldsymbol{y}[\boldsymbol{u},\boldsymbol{v}]}{\sqrt{\mathbf{1}+\mathbf{U1}[\boldsymbol{x}[\boldsymbol{u},\boldsymbol{v}]]^{\mathbf{2}}}};$$

$\mathbf{ParametricPlot}[\{\{\boldsymbol{X}[\boldsymbol{u},\boldsymbol{v}],\boldsymbol{Y}[\boldsymbol{u},\boldsymbol{v}]\}\},\{\boldsymbol{u},\mathbf{0},\boldsymbol{L}\},\{\boldsymbol{v},-\boldsymbol{h}/\mathbf{2},\boldsymbol{h}/\mathbf{2}\}]$



Следующим шагом будет показать в цвете продольные напряжения σ_x в балке. Если поперечное сечение балки прямоугольное, то σ_x вычисляются по формуле $\sigma_x(x,y)=\frac{M(x)\cdot y}{J}$, где $M(x)$ – момент инерции в сечении x балки, y – вертикальное смещение точки балки относительно нейтральной оси. Чтобы использовать эту функцию в качестве функции цвета, ее нужно привести к диапазону $[0,1]$. Для этого нужно знать минимальные и максимальные значения напряжения σ_x. В нашей задаче они, очевидно, равны напряжениям в срединном сечении балки в верхней и нижней точках $\sigma_{x\,\max/\min}=\pm\frac{M(L/2)\cdot(h/2)}{J}$, где h – высота балки.

Создаем функцию продольных напряжений S_x и вычисляем ее максимальное и минимальное значения S_{max} и S_{min}. Они используются для построения шкалы цветов напряжения справа от рисунка. Затем рисуем область изогнутой балки, раскрашенную в соответствии со значениями функции продольных напряжений S_x.

$$\boldsymbol{M}=.;\boldsymbol{M}[\mathbf{x\_}]=\frac{\boldsymbol{q}\,\boldsymbol{x}(\boldsymbol{L}-\boldsymbol{x})}{\mathbf{2}};$$

$\mathbf{Sx}[\mathbf{u\_},\mathbf{v\_}]=\boldsymbol{M}[\boldsymbol{x}[\boldsymbol{u},\boldsymbol{v}]]\boldsymbol{y}[\boldsymbol{u},\boldsymbol{v}]/\boldsymbol{J};$

$\mathbf{Smax}=\mathbf{Sx}[\boldsymbol{L}/\mathbf{2},\boldsymbol{h}/\mathbf{2}]$

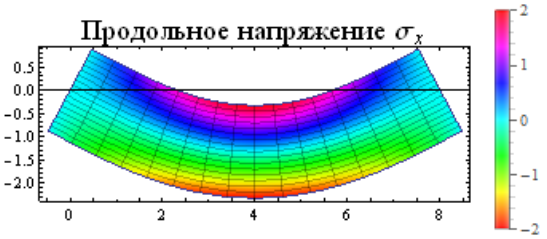
$\mathbf{Smin}=\mathbf{Sx}[\boldsymbol{L}/\mathbf{2},-\boldsymbol{h}/\mathbf{2}]$

2

-2

$\mathbf{ParametricPlot}[\{\{\boldsymbol{X}[\boldsymbol{u},\boldsymbol{v}],\boldsymbol{Y}[\boldsymbol{u},\boldsymbol{v}]\}\},\{\boldsymbol{u},\mathbf{0},\boldsymbol{L}\},\{\boldsymbol{v},-\boldsymbol{h}/\mathbf{2},\boldsymbol{h}/\mathbf{2}\},$

ColorFunction → **Function**$\left[\{x, y, u, v\}, \mathbf{Hue}\left[\mathbf{Rescale}[\mathbf{Sx}[u, v], \{-2, 2\}]\right]\right]$,
ColorFunctionScaling → **False**,
PlotLegends → **BarLegend**$\left[\{\mathbf{Hue}, \{-2, 2\}\}\right]$,
PlotLabel → **Style**["Продольное напряжение"$\boldsymbol{\sigma_x}$, **18**]]



Здесь функция `Rescale` используется для приведения диапазона [-2,2] значений функции S_x к диапазону [0,1], который требуется для функции `Hue`. Опция `ColorFunctionScaling→False` нужна для того, чтобы система не пыталась приводить аргументы функции `ColorFunction` к диапазону [0,1], что она делает по умолчанию.

Пример 6.2. Действие сосредоточенной силы на упругую полуплоскость.

Рассмотрим плоскую упругую среду, ограниченную сверху горизонтальной прямой. В некоторой точке к границе приложена вертикальная сосредоточенная сила P. Ось X направим вправо вдоль границы, а ось Y из точки приложения силы вертикально вверх. Среда расположена в полуплоскости y<0. Вертикальные напряжения Y_y в полуплоскости вычисляются по формулам [8]

$$Y_y = \frac{2P}{\pi}\frac{y^3}{\left(x^2+y^2\right)^2} \qquad (8)$$

Напряжение Yy представляет отнесенную к единице площади силу, направленную вдоль оси Y, приложенную к горизонтальной площадке (перпендикулярной направлению *y*).

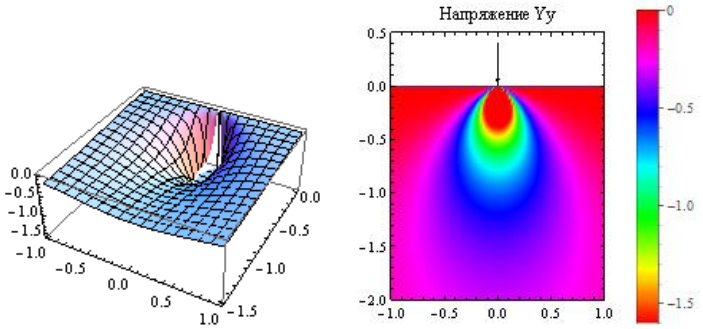
Напряжение Yy стремится к минус бесконечности в точке приложения силы. Чтобы его отобразить в цвете на полуплоскости диапазон изменения функции Yy необходимо ограничить. Для этого создадим обрезанную снизу функцию напряжений $TensYy(x, y)$, используя функцию `Piecewise`. Значение обрезания подбираем из «эстетических» соображений.

PP = **1**;
Yy[x_, y_] = $\frac{\mathbf{2PP}}{\boldsymbol{\pi}}\frac{\boldsymbol{y^3}}{(\boldsymbol{x^2}+\boldsymbol{y^2})^2}$;
TensYy[x_, y_] = **Piecewise**[{{**Yy**[x, y], **Yy**[x, y] > −1.6}}, −1.6];
Plot3D[**TensYy**[x, y], {x, −**1**, **1**}, {y, **0**, −**1.5**}] (* следующий рисунок слева *)

Теперь строим полуплоскость с окраской, соответствующей значениям функции TensYy, предварительно нормируя ее к диапазону [0, 1] (следующий рисунок справа)

p = **ParametricPlot**[{{u, v}}, {u, −**1.**, **1.**}, {v, −**2**, **0.5**},
PlotPoints → **100**, **Mesh** → **None**,

```
RegionFunction → Function[{x, y, u, v}, y < 0],
PlotRange → {{-1., 1.}, {-2, 0.5}},
ColorFunction → Function[{x, y, u, v},
        Hue[Rescale[TensYy[u, v], {-1.6, 0}]]],
ColorFunctionScaling → False,
PlotLegends → BarLegend[{Hue, {-1.6, 0}}],
Axes → None, PlotLabel → "Напряжение Yy"];
```

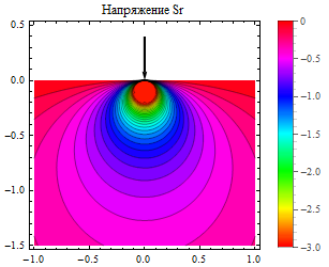


На последнем рисунке ярко красная зона (под точкой приложения нагрузки) соответствует большим отрицательным (сжимающим) напряжениям, а холодный красный (у поверхности вдали от точки приложения нагрузки) соответствует близким к нулю напряжениям. Это показывает шкала цветов, построенная справа от графика.

Известно [7], что любой элемент среды, расположенный на расстоянии r от точки приложения силы, подвергается простому сжатию в радиальном направлении. При этом компоненты напряжения определяются формулами $\sigma_r = \frac{2P}{\pi}\frac{\cos\theta}{r}$, $\sigma_\theta = \tau_{r\theta} = 0$, где θ – угол, образуемый радиус – вектором точки с вертикалью. Выберем окружность произвольного диаметра d с центром на вертикальной оси и касательную к поверхности полуплоскости. Для любой точки этой окружности имеем $d\cos\theta = r$. Отсюда, согласно выражению для напряжения σ_r получаем $\sigma_r = \frac{2P}{\pi d}$, т.е. напряжения σ_r во всех точках окружности остаются одинаковыми, за исключением точки приложения нагрузки. Проиллюстрируем этот факт графически. Для этого изобразим напряжения $\sigma_r(x,y) = \frac{2P}{\pi}\frac{\cos\theta}{r} = \frac{2Py}{\pi\left(x^2+y^2\right)}$ в области полуплоскости в виде контурного графика

```
Sr[x_, y_] = 2 PP y / (π (x^2 + y^2));
p = ContourPlot[TensSr[x, y], {x, -1, 1}, {y, -1.5, 0.5},
        RegionFunction → Function[{x, y, f}, y < 0],
        Contours → Table[pp, {pp, -3, 0, 0.1}],
        PlotRange → {-3, 0}, ClippingStyle-> Automatic,
        ColorFunction → Function[{f}, Hue[Rescale[f, {-3, 0}]]],
```

```
    ColorFunctionScaling → False, Axes → None,
    PlotLegends → Automatic];
ar = Graphics[{Thickness[0.01], Arrow[{{0, 0.4}, {0, 0}}]}];
Show[p, ar, PlotLabel → "Напряжение Sr"]
```



Кака видно, линии уровня функции $S_r(x, y)$ (напряжения σ_r) являются окружностями. Ярко красный круг под точкой приложения нагрузки представляет область обрезания, в которой $\sigma_r \le -3$.

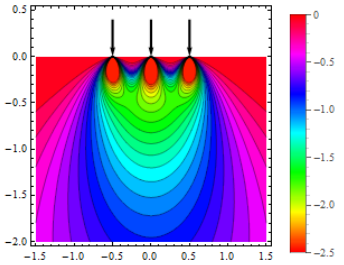
Обратите внимание на следующие особенности этого кода. Опция `PlotRange->{-3,0}` задает диапазон изменения значений функции $S_r(x, y)$ и, тем самым, определяет область отсекаемых значений. Чтобы отсекаемые зоны изображались как часть поверхности, а не как «дырки», используется опция `ClippingStyle->Automatic`. Диапазон `{-3,0}`, задаваемый опцией `PlotRange`, также используется опцией `PlotLegends->Automatic`, которая строит контрольную цветовую полосу справа от графика.

Пример 6.3. Действие нескольких сосредоточенной силы на упругую полуплоскость.

В предыдущем примере мы изобразили сжимающие напряжения Yy в бесконечной полуплоскости, возникающие под действием одной сосредоточенной силы. Поскольку уравнения теории упругости линейные, то напряжения, возникающие от действия нескольких сосредоточенных сил, будут суммой напряжений (8), возникающих от действия каждой из сил.

Пусть точки приложения трех сосредоточенных сил имеют координату $x = -0.5, 1.0, 0.5$. Изобразим суммарное Yy напряжение в полуплоскости в виде контурного графика

```
PP = 1;  Yy[x_, y_] = 2PP/π y^3/(x^2 + y^2)^2;
minYy = −2.5;  (* уровень обрезания напряжения *)
SYy[x_, y_] = Yy[x, y] + Yy[x − 0.5, y] + Yy[x + 0.5, y];
p = ContourPlot[SYy[x, y], {x, −1.5, 1.5}, {y, −2, 0.5},
      RegionFunction → Function[{x, y, f}, y < 0],
      Contours → Table[pp, {pp, minYy, 0, 0.1}],
      ColorFunction → Function[{f}, Hue[Rescale[f, {minYy, 0}]]],
      ColorFunctionScaling → False, Axes → None,
      PlotRange → {minYy, 0}, ClippingStyle → Automatic,
      PlotLegends → Automatic];
```



Опция `PlotRange->{minYy,0}` задает диапазон изменения значений функции $SY_y(x,y)$. Чтобы отсекаемые зоны изображались как часть поверхности используется опция `ClippingStyle->Automatic`. Опция `PlotLegends->Automatic` строит справа от графика шкалу цветов с диапазоном `{minYy,0}`, который был указан в опции `PlotRange`.

Пример 6.4. Напряжения в бесконечной плоской пластинке при наличии круглого отверстия.

Решим задачу о визуализации напряжений в бесконечной пластинке с отверстием, подверженной однородному растяжению величиной P (усилие, приходящееся на единицу длины). Введем полярную систему координат (r,θ), начало которой положим в центре отверстия радиуса a, а полярную ось направим вдоль усилия P. Известно [7], что напряжения в такой пластинке определяются по формулам

$$\sigma_r = \frac{P}{2}\left(1-\frac{a^2}{r^2}\right)+\frac{P}{2}\left(1+\frac{3a^4}{r^4}-\frac{4a^2}{r^2}\right)\cos 2\theta$$

$$\sigma_\theta = \frac{P}{2}\left(1+\frac{a^2}{r^2}\right)-\frac{P}{2}\left(1+\frac{3a^4}{r^4}\right)\cos 2\theta \qquad (9)$$

$$\tau_{r\theta} = -\frac{P}{2}\left(1-\frac{3a^4}{r^4}+\frac{2a^2}{r^2}\right)\sin 2\theta$$

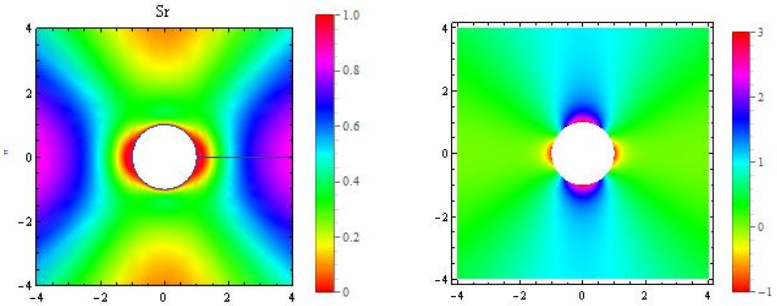
Изобразим напряжения σ_r в виде функциональной окраски области с круговым отверстием. Для этого создадим функцию $\sigma_r = S_r(r,\theta)$

```
P = 1; a = 1;
Sr[r_, v_] = P/2 (1 - a^2/r^2) + P/2 (1 + 3 a^4/r^4 - 4 a^2/r^2) Cos[2 v];
```

Из выражения для напряжения σ_r видно, что его минимальное и максимальное значения равны 0 и P. Эти величины мы используем при нормировке `ColorFunction` и задания диапазона значений для шкалы цветов.

```
ParametricPlot[{{u Cos[v], u Sin[v]}}, {u, 0, 6}, {v, 0, 2 π},
    Mesh → None, PlotPoints → 50,
    RegionFunction → Function[{x, y, u, v}, x^2 + y^2 > 1],
    PlotRange → {{-4, 4}, {-4, 4}},
    ColorFunction → Function[{x, y, u, v}, Hue[Rescale[Sr[u, v], {0, 1}]]],
    ColorFunctionScaling → False,
```

PlotLegends → BarLegend[{Hue, {0, 1}}],
Axes → None, PlotLabel → "Sr"] (* следующий рисунок слева *)



Напомним, что растяжение выполняется в горизонтальном направлении вдали от отверстия.

Напряжение σ_θ графически представим с помощью функции `DensityPlot`, предварительно записав его в декартовой системе координат (а не полярной, как в формулах (9)). Легко видеть, что $\cos 2\theta = \dfrac{x^2 - y^2}{x^2 + y^2}$. Поэтому

$$\mathbf{StC[x\_, y\_] = \frac{P}{2}\left(1 + \frac{a^2}{x^2 + y^2}\right) - \frac{P}{2}\left(1 + 3\frac{a^4}{(x^2 + y^2)^2}\right)\frac{x^2 - y^2}{x^2 + y^2};}$$

Минимальное и максимальное значение σ_θ равны –P и 3P. График плотности функции $\sigma_\theta = StC(x, y)$, изображенный на предыдущем рисунке справа, строится следующим образом

DensityPlot[StC[x, y], {x, −4, 4}, {y, −4, 4},
RegionFunction → Function[{x, y, f}, x² + y² > 1],
PlotPoints → 30, PlotRange → {−1, 3},
ColorFunction → Hue, PlotLegends → Automatic]

Нашей следующей задачей является представление продольных напряжения σ_x в области пластинки. Напряжение σ_x, заданное в декартовой системе координат, выражается через напряжения $\sigma_r, \sigma_\theta, \tau_{r\theta}$ в полярной системе по формуле $\sigma_x = \sigma_r \cos^2\theta + \sigma_\theta \sin^2\theta - \tau_{r\theta} \sin 2\theta$, где

$$\sin(2\theta) = \frac{2xy}{x^2 + y^2},\ \cos^2(\theta) = \frac{x^2}{x^2 + y^2},\ \sin^2(\theta) = \frac{y^2}{x^2 + y^2},\ r^2 = x^2 + y^2.$$

Следующий код выполняет все необходимые преобразования и использует функцию $\sigma_\theta = StC(x, y)$, определенную нами выше.

P = 1; a = 1;

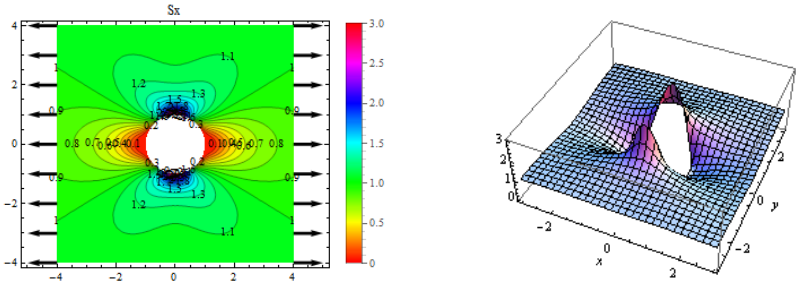
$$\mathbf{SrC[x\_, y\_] = \frac{P}{2}\left(1 - \frac{a^2}{x^2 + y^2}\right) + \frac{P}{2}\left(1 + 3\frac{a^4}{(x^2 + y^2)^2} - 4\frac{a^2}{x^2 + y^2}\right)\frac{x^2 - y^2}{x^2 + y^2};}$$

$$\mathbf{TrtC[x\_, y\_] = -\frac{P}{2}\left(1 - 3\frac{a^4}{(x^2 + y^2)^2} + 2\frac{a^2}{x^2 + y^2}\right)\frac{2xy}{x^2 + y^2};}$$

$$\mathbf{SxC[x\_, y\_] = SrC[x, y]\frac{x^2}{x^2 + y^2} + StC[x, y]\frac{y^2}{x^2 + y^2} - TrtC[x, y]\frac{2xy}{x^2 + y^2};}$$

Для построения контурного графика и шкалы цветов надо определить минимальное и максимальное значение напряжения σ_x. Они равны 0 и 3P соответственно. Тогда

```
pc = ContourPlot[SxC[x, y], {x, -4, 4}, {y, -4, 4},
        RegionFunction → Function[{x, y, f}, x^2 + y^2 > 1],
        Contours → Table[pp, {pp, 0, 3, 0.1}], ColorFunction → Hue,
        ClippingStyle → Automatic, Axes → None,  PlotLabel → Sx,
        PlotLegends → BarLegend[{Hue, {0, 3}}],
        PlotRange → {0, 3}, ContourLabels → All]
arR = Graphics[
        Table[{Thickness[0.01], Arrow[{{4, i}, {5, i}}]}, {i, -4, 4, 1}]];
arL = Graphics[
        Table[{Thickness[0.01], Arrow[{{-4, i}, {-5, i}}]}, {i, -4, 4, 1}]];
Show[pc, arR, arL, PlotRange → All, AspectRatio → Automatic]
```



Естественно, что любую функцию двух переменных можно изобразить как поверхность. Продольные напряжения, показанные на предыдущем рисунке справа, строятся как график функции двух переменных следующим кодом

```
Plot3D[SxC[x, y], {x, -3, 3}, {y, -3, 3},
   RegionFunction → Function[{x, y, f}, x^2 + y^2 > a^2],
   PlotPoints → 30, Mesh → Full, PlotRange → All]
```

Из этих графиков видно, что максимальное значения продольное напряжение σ_x достигается на концах диаметра круга, перпендикулярного к направлению растяжения, и оно в 3 раза больше постоянного напряжения P, приложенного на концах пластинки.

2.4.7 Задачи электростатики

Пример 7.1. Электростатический потенциал точечного заряда в трехмерном пространстве вычисляется по формуле [2]

$$\varphi(\mathbf{r},\mathbf{p}_0)=\varphi(x,y,z,x_0,y_0,z_0)=\frac{q}{\sqrt{(x-x_0)^2+(y-y_0)^2+(z-z_0)^2}}, \qquad (1)$$

где $\mathbf{p_0}=(x_0,y_0,z_0)$ – точка расположения заряда. Потенциал поля n зарядов вследствие суперпозиции силовых полей будет выражаться формулой

$$\varphi=\sum_{i=1}^{n}\varphi(\mathbf{r},\mathbf{p}_i). \qquad (2)$$

Изобразим потенциал поля двух единичных зарядов противоположного знака, расположенных в точках $(-1,0,0)$ и $(1,0,0)$, с помощью контурного графика. Для этого создадим функцию, реализующую формулу (2)

$$\boldsymbol{\varphi}[\mathbf{q\_}, \mathbf{p\_}, \mathbf{r\_}] := \mathbf{Sum}\left[\frac{\boldsymbol{q}[[\boldsymbol{i}]]}{\mathbf{Norm}[\boldsymbol{r} - \boldsymbol{p}[[\boldsymbol{i}]]]}, \{\boldsymbol{i}, \mathbf{Length}[\boldsymbol{q}]\}\right];$$

Здесь **q** представляет список значений зарядов, **p** является списком координат точек расположения этих зарядов, ***r*** списком имен пространственных переменных. Функция `Norm` вычисляет норму вектора/списка

$$\mathbf{Norm}[\{\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}\}]$$

$$\sqrt{\mathrm{Abs}[x]^2 + \mathrm{Abs}[y]^2 + \mathrm{Abs}[z]^2}$$

Или

$$\mathbf{Norm}[\{\boldsymbol{x_1}, \boldsymbol{y_1}, \boldsymbol{z_1}\} - \{\boldsymbol{x_2}, \boldsymbol{y_2}, \boldsymbol{z_2}\}]$$

$$\sqrt{\mathrm{Abs}[x_1 - x_2]^2 + \mathrm{Abs}[y_1 - y_2]^2 + \mathrm{Abs}[z_1 - z_2]^2}$$

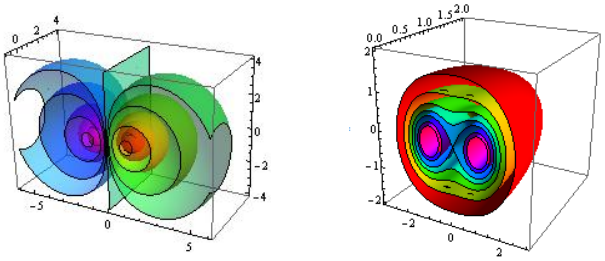
Вот, что возвращает функция $\boldsymbol{\varphi}[\mathbf{q}, \mathbf{p}, \mathbf{r}]$ для двух единичных зарядов противоположного знака, расположенных в точках $(-1,0,0)$ и $(1,0,0)$.

$$\boldsymbol{\varphi}[\{-\mathbf{1}, \mathbf{1}\}, \{\{-\mathbf{1}, \mathbf{0}, \mathbf{0}\}, \{\mathbf{1}, \mathbf{0}, \mathbf{0}\}\}, \{\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}\}]//\mathbf{TraditionalForm}$$

$$\frac{1}{\sqrt{|x-1|^2 + |y|^2 + |z|^2}} - \frac{1}{\sqrt{|x+1|^2 + |y|^2 + |z|^2}}$$

Команда построения контурного графика функции **φ** в трехмерном пространстве может иметь вид (следующий график слева)

```
ContourPlot3D[
  Evaluate[φ[{1, -1}, {{-1, 0, 0}, {1, 0, 0}}, {x, y, z}]],
  {x, -6, 6}, {y, -1, 4}, {z, -4, 4},
  Contours → {-1, -0.5, -0.25, -0.1, -0.05, 0, 0.05, 0.1, 0.25, 0.5, 1},
  ContourStyle → Table[{Opacity[0.5], Hue[i/11]}, {i, 0, 10}],
  Mesh → None, BoxRatios → {12, 5, 8}]  (* следующий рисунок слева *)
```



Для двух положительных единичных зарядов контурный график, показанный на предыдущем рисунке справа, построен следующим кодом

```
ContourPlot3D[
    Evaluate[φ[{1, 1}, {{-1, 0, 0}, {1, 0, 0}}, {x, y, z}]],
    {x, -3, 3}, {y, 0, 2}, {z, -2, 2},
    Contours → {1, 1.25, 1.5, 1.75, 2, 2.5, 3},
    ContourStyle → Table[Hue[i/7], {i, 0, 6}], Mesh → None]
```

Используя функции $\boldsymbol{\varphi}[\mathbf{q}, \mathbf{p}, \mathbf{r}]$ и `ContourPlot3D`, можно построить потенциал электростатического поля любого конечного множества точечных зарядов.

Пример 7.2. Рассмотрим распределение зарядов в пространстве, обладающее цилиндрической симметрией, т.е. рассмотрим электростатическое поле заряженной прямой. Электростатический потенциал прямой в трехмерном пространстве, совпадающей с осью Z, задается формулой

$$\varphi = 2\,q \ln \frac{1}{\rho}, \tag{3}$$

где $\rho = \sqrt{x^2 + y^2}$ и q – линейная плотность заряда вдоль линии. Решение (3) называется логарифмическим потенциалом и обладает круговой симметрией вокруг полюса в точке $\rho = 0$, в которой он обращается в бесконечность. Аналогично, потенциал однородной прямой, перпендикулярной плоскости *z*=0 и проходящей через точку (x_0, y_0), дает плоское поле и выражается формулой.

$$\varphi(\mathbf{r}, \mathbf{p}_0) = \varphi(x, y, x_0, y_0) = 2q \ln \frac{1}{\sqrt{(x - x_0)^2 + (y - y_0)^2}} \tag{4}$$

Потенциал поля n зарядов вследствие суперпозиции силовых полей будет выражаться формулой (2).

Изобразим потенциал поля двух противоположно заряженных прямых, расположенных в точках $(-1, 0)$ и $(1, 0)$, с помощью контурного графика на плоскости XY. Для этого создадим функцию, реализующую формулу (4)

```
φ[q_, p_, r_] := Sum[2q[[i]]Log[1/Norm[r - p[[i]]]], {i, Length[q]}];
```

Здесь **q** представляет список значений линейных плотностей зарядов, **p** является списком координат $\{x_i, y_i\}$ точек пересечения прямых с плоскостью z=0, ***r*** списком имен переменных (координат в плоскости), например, $\{x, y\}$.

Вот, что дает функция $\boldsymbol{\varphi}[\mathbf{q}, \mathbf{p}, \mathbf{r}]$ для двух логарифмических потенциалов с плотностями q_1, q_2, расположенных в точках (x_1, y_1) и (x_2, y_2).

```
φ[{q1, q2}, {{x1, y1}, {x2, y2}}, {x, y}]//TraditionalForm
```

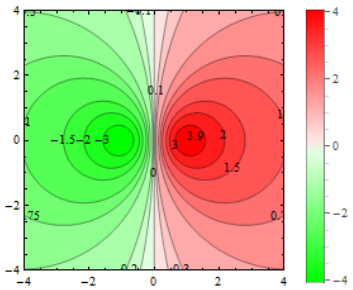
$$2q_1 \log\left(\frac{1}{\sqrt{|x - x_1|^2 + |y - y_1|^2}}\right) + 2q_2 \log\left(\frac{1}{\sqrt{|x - x_2|^2 + |y - y_2|^2}}\right)$$

Создадим список цветов для контурных кривых. Для положительных значений потенциала будем использовать красные оттенки, для отрицательных – зеленые.

```
c = Join[Table[Lighter[Green, i/9], {i, 0, 8}],
         Table[Lighter[Red, i/9], {i, 8, 0, -1}]];
```

Функция `Lighter[Color,f]` создает оттенок цвета `Color`, определяемого числом f $(0 \le f \le 1)$. Теперь строим контурный график.

```
ContourPlot[Evaluate[φ[{-1, 1}, {{-1, 0}, {1, 0}}, {x, y}]],
    {x, -4, 4}, {y, -4, 4}, PlotRangePadding → None,
    Contours → {-3, -2, -1.5, -1, -0.75, -0.5, -0.2, -0.1, 0,
                0.1, 0.2, 0.5, 0.75, 1., 1.5, 2, 3},
    ClippingStyle → Automatic, ContourShading → c,
    PlotLegends → Automatic, ContourLabels → All]
```



Ниже мы хотим смоделировать в манипуляторе поле нескольких точечных «зарядов». Но вначале мы выполним построения по шагам.

Предварительно скорректируем функцию потенциала φ так, чтобы избавиться от модулей.

$$\boldsymbol{\varphi}[\mathbf{q\_, p\_, r\_}] := \mathbf{Simplify}[\mathbf{Sum}[2\boldsymbol{q}[[\boldsymbol{i}]]\mathbf{Log}[\frac{\mathbf{1}}{\mathbf{Norm}\left[\boldsymbol{r}-\boldsymbol{p}[[\boldsymbol{i}]]\right]}], \{\boldsymbol{i}, \mathbf{Length}[\boldsymbol{q}]\}],$$
$$\mathbf{Join}[\{\mathbf{Element}[\boldsymbol{r}, \mathbf{Reals}]\}, \mathbf{Map}[\mathbf{Element}[\#, \mathbf{Reals}]\&, \boldsymbol{p}]]];$$

Тогда, например,

$\boldsymbol{\varphi}[\{\boldsymbol{q_1}, \boldsymbol{q_2}\}, \{\{\boldsymbol{x_1}, \boldsymbol{y_1}\}, \{\boldsymbol{x_2}, \boldsymbol{y_2}\}\}, \{\boldsymbol{x}, \boldsymbol{y}\}]//\mathbf{TraditionalForm}$

$q_1(-\log((x-x_1)^2+(y-y_1)^2)) - q_2\log((x-x_2)^2+(y-y_2)^2)$

или

$\boldsymbol{\varphi}[\{\boldsymbol{q_1}\}, \{\{\boldsymbol{x_1}, \boldsymbol{y_1}\}\}, \{\boldsymbol{x}, \boldsymbol{y}\}]//\mathbf{TraditionalForm}$

$q_1(-\log((x-x_1)^2+(y-y_1)^2))$

Для вычисления градиента можно использовать функцию дифференцирования D в виде

$\mathbf{Clear}[\boldsymbol{f}, \boldsymbol{x}, \boldsymbol{y}];\ \boldsymbol{D}[\boldsymbol{f}[\boldsymbol{x}, \boldsymbol{y}], \{\{\boldsymbol{x}, \boldsymbol{y}\}\}]$

$\{f^{(1,0)}[x,y], f^{(0,1)}[x,y]\}$

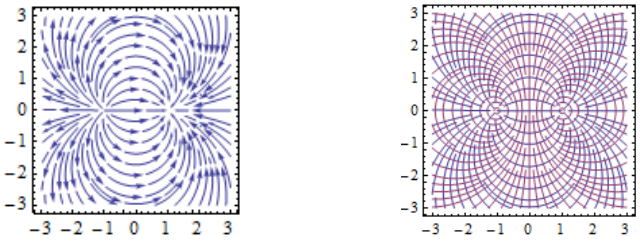
Вторым аргументом такого использования D должен быть список списков имен координат. Тогда, например

$\boldsymbol{D}[\boldsymbol{\varphi}[\{\mathbf{1}\}, \{\{\mathbf{0}, \mathbf{0}\}\}, \{\boldsymbol{x}, \boldsymbol{y}\}], \{\{\boldsymbol{x}, \boldsymbol{y}\}\}]$

$$\left\{-\frac{2x}{x^2+y^2}, -\frac{2y}{x^2+y^2}\right\}$$

В *Mathematica* есть функция `StreamPlot`, которая умеет строить линии тока векторного поля (одного или нескольких). Например силовые линии поля двух зарядов -1 и +1, расположенных в точках $(-1,0)$ и $(+1,0)$, можно построить следующим образом (следующий рисунок слева)

$\mathbf{StreamPlot}[\mathbf{Evaluate}[\boldsymbol{D}[\boldsymbol{\varphi}[\{-\mathbf{1}, \mathbf{1}\}, \{\{-\mathbf{1}, \mathbf{0}\}, \{\mathbf{1}, \mathbf{0}\}\}, \{\boldsymbol{x}, \boldsymbol{y}\}], \{\{\boldsymbol{x}, \boldsymbol{y}\}\}]],$
$\{\boldsymbol{x}, -\mathbf{3}, \mathbf{3}\}, \{\boldsymbol{y}, -\mathbf{3}, \mathbf{3}\}, \mathbf{StreamScale} \to \mathbf{0.2}]$



Кроме силовых линий мы будем строить линии уровня потенциала. Их можно строить по – разному, например, как контурный график. Мы поступим по – другому. Создадим векторное поле, ортогональное полю градиента, и построим

его линии тока. Очевидно, они будут линиями уровня потенциала. Этот способ мы выбираем из – за того, что линии тока обеих векторных полей можно построить одной функцией `StreamLine`. Например для предыдущего поля, создаваемого двумя зарядами, следующий код строит силовые линии поля и линии уровня потенциала (предыдущий рисунок справа)

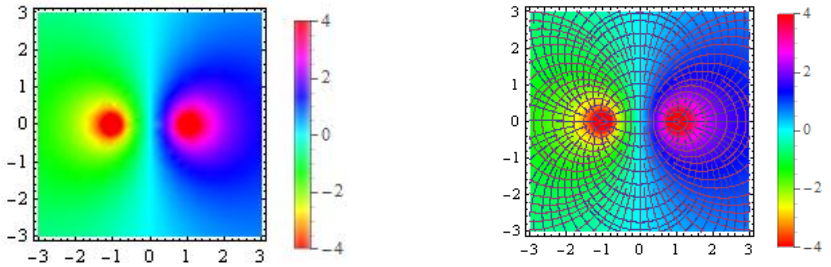
```
gd = Simplify[D[φ[{-1, 1}, {{-1, 0}, {1, 0}}, {x, y}], {{x, y}}]]
gdu = Reverse[gd * {1, -1}];
ps = StreamPlot[Evaluate[{gd, gdu}], {x, -3, 3}, {y, -3, 3},
                                    StreamScale → None]
```

$$\left\{\frac{4-4x^2+4y^2}{(1-2x+x^2+y^2)(1+2x+x^2+y^2)}, -\frac{8xy}{(1-2x+x^2+y^2)(1+2x+x^2+y^2)}\right\}$$

Здесь опция `StreamScale->None` отменяет рисование стрелок на линиях поля. Векторное поле градиента приведено сразу после кода, а векторное поле **gdu** ортогонально к полю градиента и получается перестановкой его координатных функций, перед одной из которых изменен знак.

Теперь строим график плотности потенциала (следующий рисунок слева)

```
pd=DensityPlot[
    Evaluate[φ[{-1, 1}, {{-1, 0}, {1, 0}}, {x, y}]], {x, -3, 3}, {y, -3, 3},
    ColorFunction → Hue, PlotLegends → Automatic,
    PlotRange → {-4, 4}, ClippingStyle → Automatic]
```



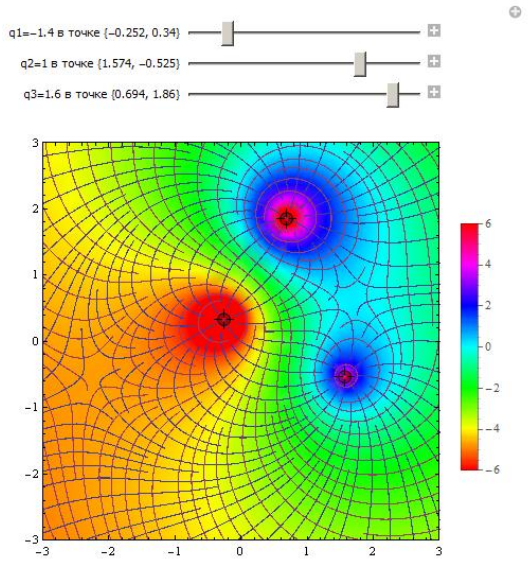
Оба графика совмещаем с помощью функции `Show`. При этом первым должен рисоваться график плотности **pd**, иначе он закроет график линий тока **ps**.

```
Show[pd, ps]   (* предыдущий рисунок справа *)
```

Наконец создадим манипулятор, в котором построим поле трех точечных «зарядов» в виде графика плотности с силовыми линиями и линиями уровня потенциала. Вот результирующий код.

```
DynamicModule[{φ, z, gd, gdu, m = 3, ps, pd},
  φ[q_, p_, r_] := Simplify[Sum[2q[[i]]Log[1/Norm[r - p[[i]]]], {i, Length[q]}],
                    Join[{Element[r, Reals]}, Map[Element[#, Reals]&, p]]];
  z[x_, y_, q_, p_] := φ[q, p, {x, y}];
  Manipulate[
    gd = Simplify[D[z[x, y, {q1, q2, q3}, p], {{x, y}}]];
    gdu = Reverse[gd * {1, -1}];
    ps = StreamPlot[Evaluate[{gd, gdu}], {x, -m, m}, {y, -m, m},
                                        StreamScale → None];
```

```
pd = DensityPlot[Evaluate[z[x, y, {q1, q2, q3}, p]],
        {x, -m, m}, {y, -m, m}, ColorFunction → Hue,
        PlotLegends → Automatic, PlotRange → {-6, 6},
        ClippingStyle → Automatic];
   Show[pd, ps, PlotRange → {{-m, m}, {-m, m}},
        PlotRangePadding → None],
{{q1, -1, Dynamic["q1 = " <> ToString[q1] <> "в точке" <>
                        ToString[p[[1]]]]}, -2, 2, 0.1},
{{q2, 1, Dynamic["q2 = " <> ToString[q2] <> "в точке" <>
                        ToString[p[[2]]]]}, -2, 2, 0.1},
{{q3, 1, Dynamic["q3 = " <> ToString[q3] <> "в точке" <>
                        ToString[p[[3]]]]}, -2, 2, 0.1},
{{p, {{-1, 0}, {1, 0}, {0, 1}}}, {-2, -2}, {2, 2}, Locator},
Paneled → False, ContinuousAction → False]
```



Основные пояснения для этого кода были выполнены выше. Код внутри функции `Manipulate` после функции `Show` выводит текстовую информацию слева от ползунков – значение заряда и его координаты, и создает три «локатора», координаты которых определяют положение зарядов. Опция **ContinuousAction → False** позволяет перерисовывать область внутри манипулятора только, когда вы отпускаете ползунок. Иначе перерисовка выполняется непрерывно при движении ползунков.

Вы можете менять значение всех трех «зарядов», перемещая ползунки. Перемещая «локаторы», вы меняете расположение «зарядов». Одно из возможных состояний манипулятора показано на предыдущем рисунке.

Литература.

1. Крауфорд Ф. Волны. М., 1974 г., 528 стр.
2. Тихонов А.Н., Самарский А.А. Уравнения математической физики. – М.: Наука, 1977. – 736с.
3. Доля П.Г. Периодическое продолжение функций и решение уравнения колебаний струны в системах символьной математики.// Вестник Харьк. нац. ун-та., - 2006.- № 733. Сер. ”Математическое моделирование.
4. Dolya P.G. Solution to the homogeneous boundary value problems of free vibrations of a finite string // Journal of Mathematical Physics, Analysis, Geometry, - 2008, vol.4, No 2 , pp. 237 – 251.
5. Тимошенко С.П. Колебания в инженерном деле. – М.: Гос. изд. физ.-мат. литературы, 1959.
6. Тимошенко С.П. Сопротивление материалов. Т.1 – М.: Наука, 1965.
7. Тимошенко С.П., Гудьер Дж. Теория упругости. – М.: Наука, 1979.
8. Филоненко Бородич М.М. Теория упругости. – М.: Гос. изд. физ.-мат. лит., 1959.
9. Снеддон И.Н., Берри Д.С. Классическая теория упругости. – М.: Гос. изд. физ.-мат. лит., 1961.

3. Реализация основных понятий математического анализа.

В системе *Mathematica* реализованы все основные понятия, используемые в математическом анализе: пределы, производные, интегралы, ряды. Предполагается, что читатель знаком с этими понятиями. В данной главе представлен обзор основных функций системы, которые выполняют операции математического анализа символьно и численно.

3.1 Вычисление пределов

Для вычисления предела используется команда `Limit`

$$\mathbf{Limit}\left[\frac{\mathbf{Sin}[x]}{x}, x \to 0\right]$$

1

Первый аргумент - функция, для которой находится предел. Второй аргумент определяет переменную и значение, к которому она приближается. Запись `x→0` читается, как "*x* стремится к нулю". Вычисление предела можно записать в традиционном виде.

$$\mathbf{Limit}[f[x], x \to x_0]//\mathbf{TraditionalForm}$$

$$\lim_{x \to x_0} f(x)$$

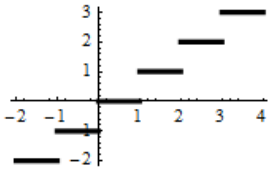
Выполним несколько примеров.

$$\mathbf{Limit}[\frac{\sqrt{1+x}-\sqrt{1-x}}{x}, x \to 0]$$

1

$$\mathbf{Limit}[\frac{(\mathbf{1}+\boldsymbol{x})^{\mathbf{2}}-\mathbf{1}}{\boldsymbol{x}},\boldsymbol{x}\to\mathbf{0}]$$
2
$$\mathbf{Limit}[\frac{\mathbf{10}^{\boldsymbol{x}}-\mathbf{1}}{\boldsymbol{x}},\boldsymbol{x}\to\mathbf{0}]$$
Log[10]
$$\mathbf{Limit}[\boldsymbol{x}\mathbf{Log}[\boldsymbol{x}],\boldsymbol{x}\to\mathbf{0}]$$
0
Можно вычислять пределы, когда переменная стремится к бесконечности
$$\mathbf{Limit}\left[\frac{(\boldsymbol{x}+\mathbf{1})^{\mathbf{2}}}{\boldsymbol{x}^{\mathbf{2}}},\boldsymbol{x}\to\infty\right]$$
1
$$\mathbf{Limit}[\frac{\mathbf{3}^{\boldsymbol{x}}-\mathbf{3}^{-\boldsymbol{x}}}{\mathbf{3}^{\boldsymbol{x}}+\mathbf{3}^{-\boldsymbol{x}}},\boldsymbol{x}\to-\infty]$$
-1
$$\mathbf{Limit}[\left(\mathbf{1}+\frac{\boldsymbol{z}}{\boldsymbol{x}}\right)^{\boldsymbol{x}},\boldsymbol{x}\to\ \mathbf{Infinity}]$$
e^{z}
$$\mathbf{Limit}[\left(\frac{\mathbf{2}\boldsymbol{n}+\boldsymbol{z}}{\mathbf{2}\boldsymbol{n}-\boldsymbol{z}}\right)^{\boldsymbol{n}},\boldsymbol{n}\to\infty]$$
e^{z}
$$\mathbf{Limit}[e^{-e^{-e^{x}}+\frac{1}{x}+x}(-\mathbf{1}+e^{e^{-e^{x}}+e^{-x}+e^{-x^{2}}}),\boldsymbol{x}\to\infty]$$
1
Можно вычислять односторонние пределы. Для этого нужно указать направление, используя опцию `Direction`. Следующая команда вычисляет левый предел (т.е. двигаясь из нуля направо или в положительном направлении)
$$\mathbf{Limit}\left[\mathbf{Tan}[\boldsymbol{x}],\boldsymbol{x}\to\frac{\boldsymbol{\pi}}{\mathbf{2}},\mathbf{Direction}\to\mathbf{1}\right]$$
∞
В следующем примере происходит приближение справа.
$$\mathbf{Limit}\left[\mathbf{Tan}[\boldsymbol{x}],\boldsymbol{x}\to\frac{\boldsymbol{\pi}}{\mathbf{2}},\mathbf{Direction}\to-\mathbf{1}\right]$$
$-\infty$
Заметим, что значение опции `Direction` определяет, куда вы хотите передвигаться, а не откуда вы идете. Таким образом, отрицательное значение указывает на движение в отрицательном направлении, т.е. приближении справа. Важен только знак, не величина (вместо единицы можно написать любое число).

Направление приближения существенно при вычислении пределов разрывных функций. Например рассмотрим функцию вычисления целой части числа (наибольшего целого, не превосходящего данное)
$$\mathbf{Plot}[\mathbf{Floor}[\boldsymbol{x}],\{\boldsymbol{x},-\mathbf{2},\mathbf{4}\}]$$



Вычислим ее пределы в точке 2.

$$\mathbf{Limit[Floor[}x\mathbf{],}x\mathbf{\to 2]}$$

2

$$\mathbf{Limit[Floor[}x\mathbf{],}x\mathbf{\to 2,Direction\to 1]}$$

1

$$\mathbf{Limit[Floor[}x\mathbf{],}x\mathbf{\to 2,Direction\to -1]}$$

2

При стремлении переменной к конечному пределу по умолчанию вычисляется предел справа (Direction $\to -1$).

Предел полезен для проверки вычисления значения функции в окрестности точек разрыва. Например вычислим значение выражения $\frac{1-\cos x^2}{x^4}$ вблизи нуля.

$$\frac{\mathbf{1-Cos[}x^{\mathbf{2}}\mathbf{]}}{x^{\mathbf{4}}}\mathbf{/.\{\{}x\mathbf{\to 0.01\},\{}x\mathbf{\to 0.00001\},\{}x\mathbf{\to 0.000010000000000000000\}\}}$$

{0.5, 0., 0.50000000000000000}

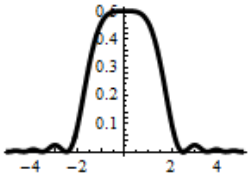
В точке *x*=0.00001 значение функции вычислено неверно. Это произошло из-за потери точности. В той же точке, но при более высокой точности задания переменной *x* (третье значение в списке) происходит вычисления значения функции с более высокой точностью. Проверка значения предлом дает 0.5

$$\mathbf{Limit}\left[\frac{\mathbf{1-Cos[}x^{\mathbf{2}}\mathbf{]}}{x^{\mathbf{4}}}\mathbf{,}x\mathbf{\to 0}\right]$$

$\frac{1}{2}$

График выражения также проясняет ситуацию.

$$\mathbf{Plot[}\frac{\mathbf{1-Cos[}x^{\mathbf{2}}\mathbf{]}}{x^{\mathbf{4}}}\mathbf{,\{}x\mathbf{,-5,5\}]}$$



Можно вычислять повторные пределы

$$\mathbf{Limit[Limit[}x^{\mathbf{2}}y^{\mathbf{2}}\mathbf{-2}xy^{\mathbf{5}}\mathbf{+3}y\mathbf{,}x\mathbf{\to 2]},y\mathbf{\to 3]}$$

-927

В некоторых случаях важен порядок вычисления пределов.

$$\mathbf{Limit}[\mathbf{Limit}\left[\frac{x^2 y}{x^2+y^2}, x \to \infty\right], y \to \infty]$$

∞

$$\mathbf{Limit}[\mathbf{Limit}\left[\frac{x^2 y}{x^2+y^2}, y \to \infty\right], x \to \infty]$$

0

Для функций нескольких переменных можно вычислять пределы по множеству

$$\mathbf{Limit}[\frac{xy}{x^2+y^2} /. x \to ty, y \to 0]$$

$$\frac{t}{1+t^2}$$

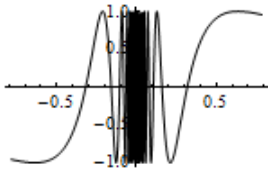
В тех случаях, когда предел не может быть вычислен, функция `Limit` возвращает интервал изменения значения функции в котором значение функции находится.

$$\mathbf{Limit}[\mathbf{Sin}[1/x], x \to 0]$$

Interval[{−1,1}]

Чтобы понять, что это значит, построим график функции

$$\mathbf{Plot}[\mathbf{Sin}\left[\frac{1}{x}\right], \{x, -\pi/4, \pi/4\}]$$



Если вычисляется предел функции, о которой нет определенной информации, то возвращается необработанный ввод.

$$\mathbf{Limit}[xf[x], x \to 0]$$

$$\text{Limit}[xf[x], x \to 0]$$

Однако имеется опция `Analytic ->True`, которая гооворит функции `Limit` делать предположения о том, что функции общего вида, о которых нет явной информации, следует считать аналитическими (в частности считать их непрерывными с непрерывными производными любого порядка). Тогда

$$\mathbf{Limit}[x\, f[x], x \to 0, \mathbf{Analytic} \to \mathbf{True}]$$

0

При вычислении пределов в функции `Limit` с помощью опции `Assumptions` можно указывать предположения относительно символьных переменных.

$$\mathbf{Limit}[x^a, x \to \infty, \mathbf{Assumptions} \to a < 0]$$

0

$$\mathbf{Limit}[x^a, x \to \infty, \mathbf{Assumptions} \to a > 0]$$

∞

$$\mathbf{Limit}[a^x/x^a, x \to \infty, \mathbf{Assumptions} \to 0 < a < 1]$$

0

Вы можете использовать `Limit` для вычисления производных «реальных» функций

$\boldsymbol{f[x\_] = x^4;}$

$\mathbf{Limit}[\frac{\boldsymbol{f[x+dx]-f[x]}}{\mathbf{dx}}, \mathbf{dx \to 0}]$

$4x^3$

$\mathbf{Limit}[\frac{\boldsymbol{f[x+2h]-2f[x+h]+f[x]}}{\boldsymbol{h^2}}, \boldsymbol{h \to 0}]$

$12x^2$

или аналитических функций общего вида

$\mathbf{Clear}[\boldsymbol{f}];$

$\mathbf{Limit}[\frac{\boldsymbol{f[x+h]-f[x]}}{\boldsymbol{h}}, \boldsymbol{h \to 0}, \mathbf{Analytic \to True}]$

$f'[x]$

$\mathbf{Limit}[\frac{\boldsymbol{f[x+2h]-2f[x+h]+f[x]}}{\boldsymbol{h^2}}, \boldsymbol{h \to 0}, \mathbf{Analytic \to True}]$

$f''[x]$

Сумма бесконечного числового ряда определяется как предел частичных сумм, например, $\sum_{k=1}^{\infty} \frac{1}{k(k+1)} = \lim_{n\to\infty} \sum_{k=1}^{n} \frac{1}{k(k+1)}$. Тогда

$\mathbf{sm = Sum}[\frac{\mathbf{1}}{\boldsymbol{k(k+1)}}, \{\boldsymbol{k}, \mathbf{1}, \boldsymbol{n}\}]$

$\mathbf{Limit}[\mathbf{sm}, \boldsymbol{n \to \infty}]$

$1 - \frac{1}{1+n}$

1

Определенный интеграл Римана является пределом интегральных сумм. Пусть это будет

$$\int_a^b f(x)\,dx = \frac{b-a}{n} \lim_{n\to\infty} \sum_{k=0}^{n-1} f\left(a + k\,\frac{b-a}{n}\right)$$

Выражение, стоящее в правой части, можно записать в виде функции

$\mathbf{rs}:= \mathbf{Function}[\{\boldsymbol{f}, \boldsymbol{x}, \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{n}\}, \frac{\boldsymbol{b-a}}{\boldsymbol{n}} \mathbf{Sum}[\boldsymbol{f}[\boldsymbol{a} + \boldsymbol{k}\frac{(\boldsymbol{b-a})}{\boldsymbol{n}}], \{\boldsymbol{k}, \mathbf{0}, \boldsymbol{n-1}\}]];$

Используя созданную функцию, можно вычислять определенный интеграл. Например, вычислим следующий интеграл $\int_0^1 x^3\,dx$.

$\mathbf{rl} = \mathbf{rs}[\#^3\&, \boldsymbol{x}, \mathbf{0}, \mathbf{1}, \boldsymbol{n}]$

$\mathbf{Limit}[\mathbf{rl}, \boldsymbol{n \to \infty}]$

$\frac{(-1+n)^2}{4n^2}$

$\frac{1}{4}$

Или вычислим интеграл $\int_0^1 e^{ax}\,dx$.

$\mathbf{rl} = \mathbf{rs}[\mathbf{Exp}[\boldsymbol{a}\#]\&, \boldsymbol{x}, \mathbf{0}, \mathbf{1}, \boldsymbol{n}];$

$\mathbf{Limit}[\mathbf{rl}, \boldsymbol{n \to \infty}]$

$$\frac{-1+e^{a}}{a}$$

Неопределенный интеграл можно вычислять как определенный с переменным верхним пределом. Например вычислим $\int_0^x t^3 d\,t$

```
rl = rs[#^3 &, t, 0, x, n];
Limit[rl, n → ∞]
```

$$\frac{x^4}{4}$$

Несобственный интеграл с бесконечными пределами является пределом определенного интеграла с конечными пределами, когда один (или оба) из пределов стремится к бесконечности. Например $\int_0^{\infty} e^{-3x}\,d\,x = \lim_{a\to\infty} \int_0^{a} e^{-3x}\,d\,x$. Тогда

```
sum0a = rs[Exp[-3#]&, x, 0, a, n];
int0a = Limit[sum0a, n → ∞];
Limit[imt0a, a → ∞]
```

$$\frac{1}{3}$$

Интеграл от неограниченных функций тоже вычисляется с использованием предельного перехода

```
Assuming[0 < ep < 1, intgr = ∫_ep^1 1/√x dx]
```

$$2-2\sqrt{\text{ep}}$$

Здесь функция `Assuming` накладывает ограничение на переменную `ep`, используемую в интеграле. В данном случае выражение (интеграл), стоящее во втором аргументе, вычисляется в предположении, что $0<ep<1$. После этого вычисляем предел

```
Limit[intgr, ep → 0]
```

2

Или

```
Assuming[0 < one < 1, intgr = ∫_0^one x/√(1 - x^2) dx];
Limit[intgr, one → 1]
```

1

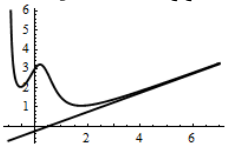
Конечно, функция `Limit` не предназначена для вычисления производных и интегралов. Для этого в системе есть более «умные» функции. С ними вы познакомитесь немного позже. Функция `Limit` нужна, например, для определения асимптот функций. Вот как это можно сделать.

```
f[x_] = (x^4 + 2x + 3)/(2x^3 + x^2 + 1);
a = Limit[f[x]/x, x → ∞]
b = Limit[f[x] - ax, x → ∞]
```

$$\frac{1}{2}$$

$$-\frac{1}{4}$$

$$\mathbf{Plot}[\{f[x], ax+b\}, \{x, -1, 7\}, \mathbf{PlotStyle} \to \{\{\mathbf{Black}, \mathbf{Thickness}[0.01]\}\}]$$



Встроенная функция `Limit` вычисляет пределы, используя аналитические методы. Кроме нее имеется внешняя функция `NLimit`, выполняющая поиск предела численно с помощью последовательности значений, приближающихся к указаному предельному значению. Результат последовательного вычисления передается в процедуру, которая аппроксимирует полученные значения и возвращает предел.

$$\mathbf{Needs}[\text{"}\mathbf{NumericalCalculus`}\text{"}]$$

(В Mathematica 5 нужно загружать пакет `<<NumericalMath`NLimit`)`

$$\mathbf{NLimit}\left[\frac{\mathbf{Sin}[x]}{x}, x->0\right]$$

1

$$\mathbf{NLimit}\left[\frac{2x^3+\mathbf{Sin}[x]}{5x^3+\mathbf{Log}[x]}, x\to\infty\right]$$

0.4

Заметим, что функция `Limit` может вернуть неправильный результат, если предельное значение указано приближенно, а не точно (числа с десятичной точкой рассматриваются как приближенные).

$$s = \mathbf{Limit}[\mathbf{Log}[1-(\mathbf{Log}[\mathbf{Exp}[z]/z-1]+\mathbf{Log}[z])/z]/z, z\to 100.]$$

$$-\infty$$

Но ответ получается правильный, если входные значения заданы точно.

$$s = \mathbf{Limit}[\mathbf{Log}[1-(\mathbf{Log}[\mathbf{Exp}[z]/z-1]+\mathbf{Log}[z])/z]/z, z\to 100]$$

$$\frac{1}{100}\text{Log}[1-\frac{1}{100}\text{Log}[-100+e^{100}]]$$

Тем не менее вычисления полученного результата с недостаточной точностью тоже неверны

$$N[s]$$

$$N[s, 20]$$

Indeterminate

-1.0000000000000000000

3.2 Дифференцирование

3.2.1 Создание и использование функций

Операция дифференцирования применяется к функциям. Поэтому здесь будет уместно напомнить некоторые основные способы создания функций.

Простейший способ состоит в выполнении команды

$\boldsymbol{y}[\mathbf{x_}] := $ *выражение*

или

$\boldsymbol{y}[\mathbf{x\_}] =$ *выражение* (без двоеточия)

Здесь *y* – имя функции (такое как Sin для функции Sin[x]). Символ подчеркивания в левой части говорит о том, что переменная *x* в правой части выражения является локальной. Знак **:=** представляет отложенное присваивание, которое выполняется в тот момент, когда происходит конкретное вычисление значения функции. Если используется знак = , то выполняется создание функции немедленно.

Общий способ создания функции пользователя состоит в использовании ключевого слова `Function`.

```
funcname = Function[x, body]
```

Эта форма предназначена для определения функции с одним формальным параметром (переменной) *x*. Здесь `funcname` имя функции, такое как Sin, Log и т.д., *body* – выражение, определяющее правило вычисления. Например

$\boldsymbol{z} = \mathbf{Function}[\boldsymbol{x}, \boldsymbol{x}^{\mathbf{2}}]$;

$\boldsymbol{z}[\mathbf{2}]$

4

Идентификатор *z*, стоящий слева, является именем функции, первое вхождение *x* в правой части обозначает имя аргумента, затем следует выражение, описывающее правило вычисления значения функции. Выражение может состоять из нескольких выражений, разделенных точкой с запятой. Результат вычисления последнего выражения в такой последовательности будет результатом вычисления функции.

$\mathbf{f1} = \mathbf{Function}[\boldsymbol{x}, \boldsymbol{g} = \boldsymbol{x}^{\mathbf{2}}; \boldsymbol{g} * \mathbf{Sin}[\boldsymbol{x}]]$;

Можно не определять имя функции при ее создании

$\mathbf{Function}[\boldsymbol{x}, \boldsymbol{x}^{\mathbf{2}}]$;

Тогда можно использовать знак % вместо имени функции (% представляет результат выполнения последней операции в системе). Например,

$\%[\boldsymbol{n}]$

n^2

Функцию можно определить без указания имен ее аргументов. В таком случае для обращения к ним следует использовать значки #1, #2 и т.д.

```
funcname=Function[body]
```

где `funcname` имя функции, *body* – выражение, определяющее правило вычисления. Например,

$\mathbf{f2} = \mathbf{Function}[\#\mathbf{1}^{\mathbf{2}}]$

$\#1^2\&$

Идентификатор #1 заменяет имя первого (и единственного) аргумента. Тогда в выражениях можно использовать `f2[x]` или вместо *x* подставлять конкретные значения `f2[2]`.

Синонимом основного определения функции является ее постфиксное определение в виде

$\mathbf{funcname} = (\#\mathbf{1}^{\mathbf{2}})\&$

$\#1^2\&$

Аргумент при определении функции обозначается # (или #1, #2 и т.д.). Само имя функции обозначается #0, ## используется для обозначения всего списка аргументов, ##n обозначает списка аргументов, начиная с n-того. Значок **&** используется вместо ключевого слова `Function` после указания тела функции. Таким образом, последнее определение эквивалентно постфиксному использованию слова `Function`.

$$\mathbf{funcname} = \overbrace{\left(\#1^2\right)}^{\&}\text{//Function}$$

Для определения функции с несколькими аргументами можно использовать форму `Function[{x1, x2,…},body]` с использованием списка формальных параметров `{x1,x2,…}`. Можно использовать и простую форму, например

$\boldsymbol{z[x\_, y\_] = x^2 + y^2;}$

$\boldsymbol{z[2, 3]}$

13

Есть еще способ создания функции с использованием символа отображения (стрелки)

$\boldsymbol{z = x \to x^3}$

$\text{Function}[x, x^3]$

Здесь стрелка особая! Чтобы правильно ее набрать надо последовательно нажать комбинацию клавиш $Esc - fn - Esc$.

Для создания функций без побочных эффектов удобно использовать ключевое слово `Module`. Имена аргументов функции указываются с символом подчеркивания в левой части операции присваивания, а в правой используется список локальных имен модуля, за которым следуют выражения

```
Module[{имя1,имя2,...},выражение1;выражение2;...]
```

Например

$\boldsymbol{f[\text{x\_}] := \text{Module}[\{t\}, t = (1 + x)\text{\^{}}3; t = \text{Expand}[t]]}$

$\boldsymbol{f[s]}$

$1 + 3s + 3s^2 + s^3$

Имена переменных, использованных в первом списке справа, будут локальными переменными модуля и не будут конфликтовать с такими же именами в глобальной области рабочего пространства системы. При этом они могут рассматриваться не только как имена переменных, но и как имена локальных функций. С использование функции `Module` могут конструироваться функции многих переменных

В *Mathematica* есть несколько способов применить функцию к выражению. Это обычные квадратные скобки f[x], префикс f@x и постфикс x // f. Например

$\{\mathbf{Sin}\left[\frac{\boldsymbol{\pi}}{\mathbf{6}}\right],\ \mathbf{Sin@}\frac{\boldsymbol{\pi}}{\mathbf{4}},\ \frac{\boldsymbol{\pi}}{\mathbf{3}}\mathbf{//Sin}\}$

$\left\{\frac{1}{2},\frac{1}{\sqrt{2}},\frac{\sqrt{3}}{2}\right\}$

Префиксное и постфиксное использование имени функции используется для функций одной переменной. Для функции двух переменных возможно инфиксное (между аргументами) использование имени функции. Например

$\boldsymbol{f}[\mathbf{x\_},\mathbf{y\_}] = \boldsymbol{x} + \boldsymbol{y}^{\mathbf{2}}$

$2 \sim f \sim 3$

11

или

$1 \sim f \sim 2 \sim f \sim 3$

14

3.2.2 Вычисление производных и дифференциалов

Любую стандартную функцию или функцию пользователя, созданную одним из приведенных выше способов можно дифференцировать. Для вычисления производны в символьной форме используются функции `Derivative` и `D`.

Функция `Derivative[n][f]` вычисляет n–ю производную функции `f` одного аргумента. Например

$\mathbf{Derivative}[\mathbf{1}][\boldsymbol{f}]$

f'

Вместо использования полного имени функции `Derivative` можно писать обычное обозначение со штрихом. Так записи f', f'' эквивалентны Derivative[1][f] и Derivative[2][f]. Например

$\boldsymbol{y}[\mathbf{x\_}] = \boldsymbol{x}^{\mathbf{3}};$

$\boldsymbol{y}'[\boldsymbol{x}]$

$\boldsymbol{y}''[\boldsymbol{x}]$

$3x^2$

$6x$

Функция `Derivative` применяется к функции (не к выражению) и возвращает функцию. Например

$\boldsymbol{f}[\mathbf{x\_}] := \boldsymbol{x}^{\mathbf{2}}\mathbf{Sin}[\boldsymbol{x}]$

$\boldsymbol{f}'$

$2\text{Sin}[\#1]\#1 + \text{Cos}[\#1]\#1^2\&$

Результат команды f' является функцией. Поэтому допустима запись

$\boldsymbol{f}'[\boldsymbol{x}]$

$x^2\text{Cos}[x] + 2x\text{Sin}[x]$

и

$f'[\pi/2]$

π

При использовании штрихов имеются некоторые особенности, вытекающие из операторной природы функции `Derivative`. Следующая форма команды дифференцирования не работает.

$(x^2 + \text{Sin}[x])'$

$(x^2 + \text{Sin}[x])'$

Однако работает

$((\#^2 + \text{Sin}[\#])\&)'[x]$

$2x + \text{Cos}[x]$

Штрих должен стоять после функции как в последнем примере или после имени функции как в примерах до этого, а не после выражения! Поэтому можно писать

Cos'

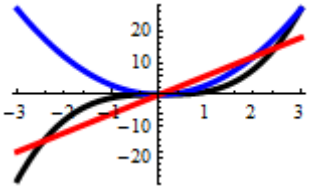
$-\text{Sin}[\#1]\&$

$\text{Cos}'[x]$

$-\text{Sin}[x]$

Функция `Derivative` удобна при построения графиков производных.

$\text{Plot}[\{y[x], y'[x], y''[x]\}, \{x, -3, 3\}, \text{PlotStyle} \to \{\text{Black}, \text{Blue}, \text{Red}\}]$



Общая форма функции Derivative[n1, n2, ...][f] вычисляет производную f по первой переменной n_1 раз, n_2 раз по второй переменной и т.д.

$g[\text{x\_}, \text{y\_}] = \text{Sin}[xy];$

$\text{Derivative}[2, 3][g][x, y]$

$-6x\text{Cos}[xy] + x^3y^2\text{Cos}[xy] + 6x^2y\text{Sin}[xy]$

Можно использовать запись Derivative[-n][f], которая представляет неопределенный интеграл n-го порядка

$y[\text{x\_}] = x^3;$

$\text{Derivative}[-1][y]$

$\frac{\#1^4}{4}\&$

$\text{Derivative}[-1][y][x]$

$\frac{x^4}{4}$

$\text{Derivative}[-2][y][x]$

$\frac{x^5}{20}$

Функция `D` используется для дифференцирования выражений. В формате `D[f,x]` она вычисляют частную производную выражения f по переменной x.

$D[x^n, x]$

$n\,x^{-1+n}$

В формате `D[f,{x,n}]` вычисляется частная производная n-го порядка выражения f по переменной *x*.

$\boldsymbol{D[x^n,\{x,4\}]}$

$(-3+n)(-2+n)(-1+n)nx^{-4+n}$

В формате D[f,x_1,x_2,…] вычисляется смешанная производная $\dfrac{\partial^n f(x_1,x_2,...)}{\partial x_1\,\partial x_2...}$. При этом дифференцирование выполняется сначала по x_1, потом по x_2 и т.д.

$\boldsymbol{D[(x^2+y^2)^3,x,y]}$

$24\,x\,y\,(x^2+y^2)$

$\boldsymbol{f[x\_,y\_]:=\mathbf{Sin}[xy]};$

$\boldsymbol{D[f[x,y],x,\{y,2\}]}$

$-x^2y\mathrm{Cos}[xy]-2x\mathrm{Sin}[xy]$

Команда `D[f,{{x1,x2,…}}]` вычисляет вектор производных $\left\{\dfrac{\partial f}{\partial x_1},\dfrac{\partial f}{\partial x_2},...\right\}$ скалярной функции `f`.

$\boldsymbol{D[f[x,y],\{\{x,y\}\}]}$

$\{f^{(1,0)}[x,y],f^{(0,1)}[x,y]\}$

$\boldsymbol{D[(x^2+y^2)^2,\{\{x,y\}\}]}$

$\{4x(x^2+y^2),4y(x^2+y^2)\}$

Запись `D[f,x]` может быть заменена на $\partial_x f$. Для ввода нужно набрать `Esc-pd-Esc`, затем `Ctrl-_` (подчеркивание), затем ввести *x*.

$\boldsymbol{\partial_x\mathbf{Sin}[x]}$

$\mathrm{Cos}[x]$

$\boldsymbol{\partial_x(a\,x^2)}$

$2a\,x$

Запись `D[f,x,y]` может быть записана в виде $\partial_{x,y} f$. Вводить надо также как предыдущий символ $\partial_x f$. При этом запятую нужно вводить, но она может быть невидимой. Для ввода невидимой запятой нужно набрать комбинацию клавиш `Esc-,(запятая)-Esc`.

$\boldsymbol{\partial_{x\,y}(x^2+y^2)^3}$ (* невидимая запятая *)

$24\,x\,y\,(x^2+y^2)$

$\boldsymbol{\partial_{x,x,y}\mathbf{Sin}[xy]}$ (* видимые запятые *)

$-xy^2\mathrm{Cos}[xy]-2y\mathrm{Sin}[xy]$

Построим для примера таблицу производных функции

$\boldsymbol{f[x\_]=x^6+\mathbf{Sin}[x]};$

$\boldsymbol{\mathbf{Grid}[\mathbf{Table}[\{\mathbf{ToString}[i]<>"-я",\partial_{\{x,i\}}f[x]\},\{i,8\}],\mathbf{Frame}\to\mathbf{All}]}$

1-я	$6\,x^5 + \mathrm{Cos}[x]$
2-я	$30\,x^4 - \mathrm{Sin}[x]$
3-я	$120\,x^3 - \mathrm{Cos}[x]$
4-я	$360\,x^2 + \mathrm{Sin}[x]$
5-я	$720\,x + \mathrm{Cos}[x]$
6-я	$720 - \mathrm{Sin}[x]$
7-я	$-\mathrm{Cos}[x]$
8-я	$\mathrm{Sin}[x]$

Вы можете дифференцировать неопределенные функции. Вот, например, правило дифференцирования сложной функции

$\boldsymbol{\partial_x f[g[x]]}$

$f'[g[x]]g'[x]$

Вот формула дифференцирования произведения трех функций

$\boldsymbol{\partial_x(u[x]v[x]w[x])}$

$v[x]w[x]u'[x] + u[x]w[x]v'[x] + u[x]v[x]w'[x]$

Если какая – либо переменная неявно зависит от переменной дифференцирования, то это можно указать с помощью опции `NonConstants`. Например,

$\boldsymbol{D[x^3 + y^3, x, \mathrm{NonConstants} \to y]}$

$3x^2 + 3y^2 D[y, x, \mathrm{NonConstants} \to \{y\}]$

Эту опцию можно использовать, чтобы имитировать дифференцирование функции, заданной неявно

$\boldsymbol{\mathrm{dd} = D[x^3 + y^3 == 1, x, \mathrm{NonConstants} \to y]}$

$3x^2 + 3y^2 D[y, x, \mathrm{NonConstants} \to \{y\}] == 0$

$\boldsymbol{\mathrm{dp} = \mathrm{dd}/.\{D[y, x, \mathrm{NonConstants} \to \{y\}] \to y'\}}$

$3x^2 + 3y^2 y' == 0$

$\boldsymbol{\mathrm{Solve}[\mathrm{dp}, y']}$

$\{\{y' \to -\frac{x^2}{y^2}\}\}$

Функция `Dt[f]` вычисляет полный дифференциал выражения f.

$\boldsymbol{\mathrm{Dt}[x^2 y^3]}$

$2xy^3\mathrm{Dt}[x] + 3x^2y^2\mathrm{Dt}[y]$

Функция `Dt[f,x]` вычисляет полную производную df/dx выражения f, полагая, что другие параметры могут зависеть от переменной x.

$\boldsymbol{\mathrm{Dt}[x^n, x]}$

$x^n\left(\frac{n}{x} + \mathrm{Dt}[n, x]\mathrm{Log}[x]\right)$

$\boldsymbol{\mathrm{Dt}[\mathrm{Sin}[x\,y], x]}$

$\mathrm{Cos}[xy](y + x\mathrm{Dt}[y, x])$

Можно сказать, что команда `D[f,x]` аналогична записи $\partial f/\partial x$, а команда `Dt[f,x]` аналогична записи df/dx.

Вторая полная производная выглядит несколько сложнее.

$\boldsymbol{\mathrm{Dt}[x^2 y^3, \{x, 2\}]}$

$2y^3 + 12xy^2\mathrm{Dt}[y, x] + x^2(6y\mathrm{Dt}[y, x]^2 + 3y^2\mathrm{Dt}[y, \{x, 2\}])$

Если в определении функции содержатся какие – либо константы, то их производные равняются нулю. Функции `Dt` это можно указать с помощью опции `Constants`, в значении которой перечислить имена этих констант. Например,

Dt[*a x y* + Sin[*k x*], *x*, Constants → {*a*, *k*}]

a y + *k* Cos[*k x*] + *a x* Dt[*y*, *x*, Constants → {*a*, *k*}]

Полная производная функции одной переменной совпадает с ее частной производной

Dt[Sin[*x*], *x*] == *D*[Sin[*x*], *x*]

`True`

Используя приведенные функции, можно решать классические задачи математического анализа. Например, можно построить частичную сумму ряда Тейлора

kf = Table[*D*[Log[1 + *x*], {*x*, *n*}]/*n*!, {*n*, 0, 8}]/. {*x* → 0}

$\{0,1,-\frac{1}{2},\frac{1}{3},-\frac{1}{4},\frac{1}{5},-\frac{1}{6},\frac{1}{7},-\frac{1}{8}\}$

Sum[kf[[*i*]] *x*^(*i*−1), {*i*, 1, 8}]

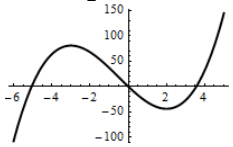
$x-\frac{x^2}{2}+\frac{x^3}{3}-\frac{x^4}{4}+\frac{x^5}{5}-\frac{x^6}{6}+\frac{x^7}{7}$

Можно находить точки экстремума

Clear[*f*]

f*[x_] = 2 *x*^3 + 3*x*^2 − 36*x

Plot[*f*[*x*], {*x*, −6, 5}, PlotRange → All]



Solve[*D*[*f*[*x*], *x*] == 0, *x*]

{{*x* → −3}, {*x* → 2}}

И точки перегиба

Solve[*D*[*f*[*x*], {*x*, 2}] == 0, *x*]

$\{\{x\to-\frac{1}{2}\}\}$

Использование производных иногда полезно для доказательства функциональных тождеств. Хорошо известен следующий факт. Если две функции $f(x)$ и $g(x)$ определены и непрерывны в промежутке $\aleph$ и внутри него имеют конечные производные $f'(x), g'(x)$, причем $f'(x)=g'(x)$ внутри $\aleph$, то эти функции во всем промежутке $\aleph$ разнятся лишь на постоянную: $f(x)=g(x)+C\ \ (C=const)$.

Пример. Рассмотрим две функции *arctg x* и $\arcsin\frac{x}{\sqrt{1+x^2}}\quad(-\infty<x<\infty)$.

Вычислим их производные.

f1[x_] = ArcTan[*x*];

$$\mathbf{f2[x\_] = ArcSin[\frac{x}{\sqrt{1+x^2}}]};$$

$$\mathbf{f1'[x]}$$

$$\frac{1}{1+x^2}$$

$$\mathbf{Simplify[f2'[x], Assumptions :\to x \in Reals]}$$

$$\frac{1}{1+x^2}$$

Производные совпадают во всем промежутке $-\infty < x < \infty$ и, следовательно, эти функции во всем промежутке разнятся на постоянную, которую можно определить в произвольной точке, например, при $x = 0$. Имеем

$$\mathbf{Const = f1[0] - f2[0]}$$

0

Т.о. $arctg\, x = \arcsin\frac{x}{\sqrt{1+x^2}} \quad (-\infty < x < \infty)$.

□

Для численного определения значения производной в точке предназначена функция ND. Она находится в пакете расширений `NumericalCalculus` (в *Mathematica* 5 в пакете `<<NumericalMath`NLimit`).

```
Needs["NumericalCalculus`"]
```

В формате `ND[expr,x,x0]` функция выполняет численную апроксимацию первой производной по x в точке x_0.

```
ND[Exp[Sin[x]], x, 2]
```

−1.03312

В формате `ND[expr,{x,n},x0]` функция выполняет численную апроксимацию n – ой производной в точке x_0.

```
ND[xSin[x], {x, 2}, 0]
```

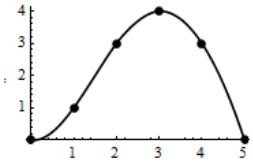
2.

`ND` полезна при вычислении производных функций, заданных численно.

```
points = {{0, 0}, {1, 1}, {2, 3}, {3, 4}, {4, 3}, {5, 0}};
ifun = Interpolation[points]
ifun[2.5]
Plot[ifun[x], {x, 0, 5}, Epilog → {PointSize[0.04], Map[Point, points]}]
```

InterpolatingFunction[{{0,5}},"<>"]

3.6875



Функция `Interpolation` строит интерполяционную функцию, определяемую списком **points**. Используя функцию `ND`, можно вычислять значение производной в конкретных точках.

```
ND[ifun[x], x, 1]
```

1.83333

Однако, объект InterpolatingFunction, созданный функцией `Interpolation` является «настоящей» функцией и его можно дифференцировать символьно.

```
D[ifun[x], x]/. x → 1.
1.83333
```

Производную в точке можно было бы вычислить по – другому так

```
ifun'[1.]
1.83333
```

Функция `ND`, в отличие от `D`, умеет вычислять односторонние производные. Подробнее с ней вы можете познакомиться по справочной системе.

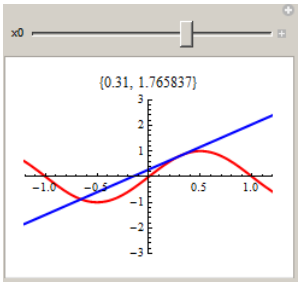
3.2.3 Геометрические приложения производных

Из курсов математического анализа известно, что значение производной в точке рано тангенсу наклона касательной к графику функции в этой точке. Уравнение касательной к кривой, заданной явно $y = f(x)$, в точке x_0 имеет вид

$$y = f(x_0) + f'(x_0)(x - x_0)$$

Пример. Касательная к графику функции $y = \sin \pi x$.

```
DynamicModule[{f, f1},
  f[x_] := Sin[πx];
  f1[x_, x0_] := f[x0] + f'[x0](x - x0);
  Manipulate[
    Plot[{f[x], f1[x, x0]}, {x, -1.5, 1.5},
      PlotRange → {{-1.2, 1.2}, {-3, 3}},
      PlotLabel → {x0, f'[x0]},
      PlotStyle → {{Red, Thickness[0.01]}, {Blue, Thickness[0.01]}}],
  {{x0, 0}, -1, 1}]]
```



Перемещая ползунок, вы будете менять точку касания x_0 и, соответственно будет перерисовываться касательная, а в заголовке графика будет отображаться текущее значение x_0 точки касания и значение производной в этой точке.

□

Пусть кривая задана неявным уравнением $F(x, y) = 0$ и точка (x_0, y_0) принадлежит кривой. Уравнение касательной к кривой в этой точке имеет вид

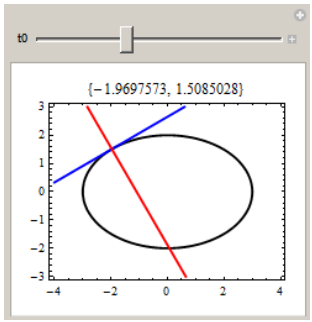
$$F'_x(x_0, y_0)(x - x_0) + F'_y(x_0, y_0)(y - y_0) = 0,$$

а уравнение нормали

$$F'_x(x_0, y_0)(y - y_0) - F'_y(x_0, y_0)(x - x_0) = 0.$$

Пример. Касательная и нормаль к эллипсу, заданному неявным уравнение

```
DynamicModule[{a = 3, b = 2, t0, x0, y0, eq1, eq2, eq3, pc1, pc2},
  eq1[x_, y_] = x^2/a^2 + y^2/b^2 - 1;
  eq2[x0_, y0_] = Derivative[1, 0][eq1][x0, y0](x - x0) +
                  Derivative[0, 1][eq1][x0, y0](y - y0);
  eq3[x0_, y0_] = Derivative[1, 0][eq1][x0, y0](y - y0) -
                  Derivative[0, 1][eq1][x0, y0](x - x0);
  pc1 = ContourPlot[eq1[x, y] == 0,
                  {x, -a - 1, a + 1}, {y, -b - 1, b + 1},
                  ContourStyle → {Black, Thickness[0.01]}];
  Manipulate[ x0 = aCos[t0]; y0 = bSin[t0];
    pc2 = ContourPlot[{eq2[x0, y0] == 0, eq3[x0, y0] == 0},
                  {x, -a - 1, a + 1}, {y, -b - 1, b + 1},
                  ContourStyle → {{Blue, Thickness[0.01]},
                                  {Red, Thickness[0.01]}}];
    Show[pc1, pc2, PlotLabel → {x0, y0}, AspectRatio → Automatic],
  {{t0, π/4}, 0, 2π}]]
```



Положение ползунка определяет точку на эллипсе, в которой строятся касательная и нормаль. В заголовке графика отображаются текущие координаты точки.

□

Единичный касательный вектор к плоской кривой, заданной параметрически $x = x(t)$, $y = y(t)$, в точке $t = t_0$ равен

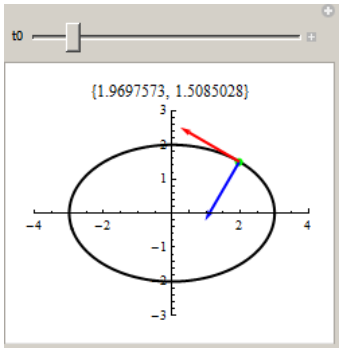
$$\mathbf{T} = (T_x, T_y) = \left(\frac{x'(t_0)}{\sqrt{(x'(t_0))^2 + (y'(t_0))^2}}, \frac{y'(t_0)}{\sqrt{(x'(t_0))^2 + (y'(t_0))^2}} \right).$$

Тогда касательная к кривой в точке $t = t_0$ может быть представлена уравнениями $x(t) = x(t_0) + T_x \cdot t$; $y(t) = y(t_0) + T_y \cdot t$.

Единичный вектор нормали к кривой в точке $t = t_0$ равен $\mathbf{N} = (-T_y, T_x)$. Тогда уравнение нормали в этой точке можно записать в виде $x(t) = x(t_0) - T_y \cdot t$; $y(t) = x(t_0) + T_x \cdot t$.

Пример. Касательная и нормаль к эллипсу, заданному параметрически.

```
DynamicModule[{a = 3, b = 2, t0, tx, ty, x, y, xt, yt, xn, yn, pe, pa, pp},
  {x[t_], y[t_]} = {aCos[t], bSin[t]};   (* радиус-вектор *)
  {tx[t_], ty[t_]} = {x'[t], y'[t]} / Sqrt[x'[t]^2 + y'[t]^2];   (* единичный вектор касательной *)
  xt[t_, t0_] = x[t0] + tx[t0] t;   (* уравнение касательной *)
  yt[t_, t0_] = y[t0] + ty[t0] t;
  xn[t_, t0_] = x[t0] - ty[t0] t;   (* уравнение нормали *)
  yn[t_, t0_] = y[t0] + tx[t0] t;
  Manipulate[
    pe = ParametricPlot[{x[t], y[t]}, {t, 0, 2 π}];
    pp = Graphics[{Green, PointSize[0.03], Point[{x[t0], y[t0]}]}];
    pa = Graphics[{
      {Blue, Thickness[0.01], Arrow[{{x[t0], y[t0]}, {xn[2, t0], yn[2, t0]}}]},
      {Red, Thickness[0.01], Arrow[{{x[t0], y[t0]}, {xt[2, t0], yt[2, t0]}}]}}];
    Show[pe, pp, pa, PlotRange → {{-4, 4}, {-3, 3}},
                     PlotLabel → {x[t0], y[t0]}],
  {t0, 0, 2 π}]]
```



Перемещая ползунок, вы будете менять точку кривой и, соответственно будут перерисовываться вектор касательной и нормали, а в заголовке графика будут отображаться текущие значения (x_0, y_0) координат точки.

□

Единичный касательный вектор к пространственной кривой, заданной параметрически $\mathbf{r}(t) = (x(t), y(t), z(t))$, в точке $t = t_0$ равен $\mathbf{T} = (T_x, T_y, T_z) = \dfrac{\mathbf{r}'(t_0)}{\|\mathbf{r}'(t_0)\|}$.

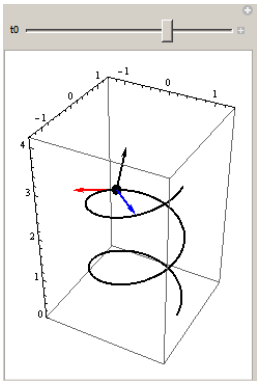
Единичный вектор главной нормали равен $\mathbf{N} = (N_x, N_y, N_z) = \dfrac{\mathbf{r}''(t_0)}{\|\mathbf{r}''(t_0)\|}$.

Единичный вектор бинормали равен $\mathbf{B} = (B_x, B_y, B_z) = [\mathbf{T}, \mathbf{N}]$. Тогда векторные уравнения касательной, главной нормали и бинормали будут иметь вид: $\mathbf{r}_t(t) = \mathbf{r}(t_0) + \mathbf{T} \cdot t$, $\mathbf{r}_n(t) = \mathbf{r}(t_0) + \mathbf{N} \cdot t$, $\mathbf{r}_b(t) = \mathbf{r}(t_0) + \mathbf{B} \cdot t$.

Пример. Репер Френе.

```
DynamicModule[{a = 1, b = 1, m = 4, tmax = 4 π, db = 0.4,
                                   t0, r, tr, nr, br, rt, rn, rb, pc, pa, pp},
  r[t_] = {aCos[t], bSin[t], t/m};   (* радиус-вектор *)
```

```
tr[t_] = Normalize[r'[t]];     (* единичный вектор касательной *)
nr[t_] = Normalize[r''[t]];    (* единичный вектор главной нормали *)
br[t_] = Simplify[Cross[tr[t], nr[t]]]; (* единичный вектор бинормали *)
rt[t_, t0_] = r[t0] + tr[t0]t;     (* уравнение касательной *)
rn[t_, t0_] = r[t0] + nr[t0]t;     (* уравнение главной нормали *)
rb[t_, t0_] = r[t0] + br[t0]t;     (* уравнение бинормали *)
Manipulate[
  pc = ParametricPlot3D[r[t], {t, 0, tmax}];
  pp = Graphics3D[{PointSize[0.03], Point[r[t0]]}];
  pa = Graphics3D[{
        {Blue, Arrow[{r[t0], rn[1, t0]}]},
        {Red, Arrow[{r[t0], rt[1, t0]}]},
        {Black, Arrow[{r[t0], rb[1, t0]}]}}];
  Show[pc, pp, pa, BoxRatios → {2a, 2b, tmax/m},
        PlotRange → {{-a - db, a + db}, {-b - db, b + db}, {0, 4.2}}],
{{t0, π}, 0, tmax}]]
```



Перемещая ползунок, вы будете менять точку кривой и соответственно будут перерисовываться вектора репера Френе.

Обратите внимание, что в приведенном коде вектор – функции **r[t], tr[t]** и другие являются списками из трех скалярных функций. Функция `Normalize` преобразует вектор в вектор единичной длины.

□

Для поверхности, заданной неявным уравнением $F(x,y,z)=0$, уравнение касательной плоскости в точке (x_0,y_0,z_0) имеет вид

$$F'_x(x_0,y_0,z_0)(x-x_0)+F'_y(x_0,y_0,z_0)(y-y_0)+F'_z(x_0,y_0,z_0)(z-z_0)=0$$

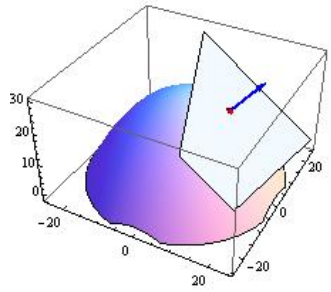
Уравнение нормали к поверхности в этой точке

$$\frac{x-x_0}{F'_x(x_0,y_0,z_0)}=\frac{y-y_0}{F'_y(x_0,y_0,z_0)}=\frac{z-z_0}{F'_z(x_0,y_0,z_0)}$$

Пример. Касательная плоскость и вектор нормали к эллиптическому параболоиду

```
eq1 =.; a = 5; b = 5;
eq1[x_, y_, z_] = x^2/a^2 + y^2/b^2 + z - 25;
```

```
x0 = 10; y0 = 4; z0 = 25 - x0^2/a^2 - y0^2/b^2;
FX = Derivative[1, 0, 0][eq1][x0, y0, z0];
FY = Derivative[0, 1, 0][eq1][x0, y0, z0];
FZ = Derivative[0, 0, 1][eq1][x0, y0, z0];
eq2 = FX(x - x0) + FY(y - y0) + FZ(z - z0);
n[t_] = {x0, y0, z0} + t {FX, FY, FZ};
pa = Graphics3D[{Blue, Thickness[0.01], Arrow[{n[0], n[10]}]}];
ps = ContourPlot3D[eq1[x, y, z] == 0,
        {x, -5a, 5a}, {y, -5b, 5b}, {z, -2, 28}, Mesh → None];
pp = ContourPlot3D[eq2 == 0,
        {x, -5a, 5a}, {y, -5b, 5b}, {z, -2, 28}, Mesh → None];
pt = Graphics3D[{Red, Sphere[{x0, y0, z0}, 1]}];
Show[ps, pp, pt, pa, PlotRange → All, BoxRatios → {50, 50, 30}]
```



□

Можно построить касательные плоскости и нормали к поверхностям, заданным явно $z = f(x, y)$ или заданным параметрически $x = x(u, v)$, $y = y(u, v)$, $z = z(u, z)$. Предоставляем читателю возможность самостоятельно рассмотреть эти примеры.

3.3 Интегрирование

3.3.1 Символьное вычисление интегралов

Для вычисления неопределенного интеграла $\int f(x) dx$ используется функция `Integrate[f,x]`.

```
Integrate[Cos[x]^2, x]
```

$$\frac{x}{2} + \frac{1}{4}\text{Sin}[2x]$$

Обратите внимание, что константа интегрирования не добавляется.

Вместо имени функции можно использовать классическое обозначение ∫.

```
∫ x^2 𝕕x
```

$$\frac{x^3}{3}$$

Трафарет ввода знака интеграла можно найти на панели специальных символов «Basic Math Input». Можно использовать также специальные комбинации клавиш. Для ввода «крючка» интеграла наберите `Esc-int-Esc`, затем введите

подынтегральное выражение, затем специальный значок дифференциала Esc–dd–Esc и имя переменной интегрирования.

$$\int \mathbf{Sqrt}[x + \mathbf{Sqrt}[x]]\, \mathbb{d}x$$

$$\frac{1}{12}\sqrt{\sqrt{x} + x}(-3 + 2\sqrt{x} + 8x) + \frac{1}{8}\mathrm{Log}[1 + 2\sqrt{x} + 2\sqrt{\sqrt{x} + x}]$$

Можно интегрировать выражения, содержащие более чем одну переменную; другие переменные будут трактоваться, как константы

$$\int a\, x^2\, \mathbb{d}x$$

$$\frac{a\, x^3}{3}$$

Можно вычислять интеграл от интеграла

$$\mathbf{Integrate}[x^2 y^3, x, y]$$

$$\frac{x^3 y^4}{12}$$

или по – другому

$$\int\int x^2 y^3\, \mathbb{d}y\, \mathbb{d}x$$

$$\frac{x^3 y^4}{12}$$

Mathematica умеет интегрировать рациональные, тригонометрические и много других функций

$$\int \frac{x^5}{x^3 + 3x^2 + 7x}\, \mathbb{d}x$$

$$2x - \frac{3x^2}{2} + \frac{x^3}{3} - \frac{73\mathrm{ArcTan}[\frac{3 + 2x}{\sqrt{19}}]}{\sqrt{19}} + \frac{15}{2}\mathrm{Log}[7 + 3x + x^2]$$

Она «знает» много интегралов, которые могут быть определены в терминах специальных функций

$$\int \mathbf{Exp}[-x^2]\, \mathbb{d}x$$

$$\frac{1}{2}\sqrt{\pi}\mathrm{Erf}[x]$$

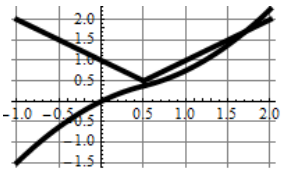
$$\int \frac{\mathbf{Sin}[x^2]}{x^5}\, \mathbb{d}x$$

$$-\frac{\mathrm{Cos}[x^2]}{4x^2} - \frac{\mathrm{Sin}[x^2]}{4x^4} - \frac{\mathrm{SinIntegral}[x^2]}{4}$$

Mathematica умеет интегрировать кусочные функции

```
Grid[{{
   px = Integrate[Max[x, 1 - x], x],
   Plot[{Max[x, 1 - x], px}, {x, -1, 2},
         PlotStyle → {{Black, Thickness[0.02]}},
         GridLines → Automatic]}}]
```

$$\begin{cases} x - \frac{x^2}{2} & x \le \frac{1}{2} \\ \frac{1}{4} + \frac{x^2}{2} & \text{True} \end{cases}$$

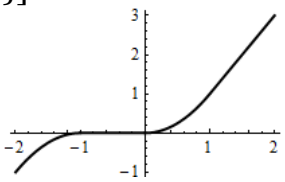


Grid[{{

$y[x\_] = \mathbf{Piecewise}\left[\left\{\{-(x+1)^2, x < -1\}, \{0, x < 0\}, \{x^2, x < 1\}\right\}, 2x - 1\right]$,

Plot[$y[x]$, {x, -2, 2}]}}]

$$\begin{cases} -(1+x)^2 & x < -1 \\ 0 & x < 0 \\ x^2 & x < 1 \\ -1 + 2x & \text{True} \end{cases}$$

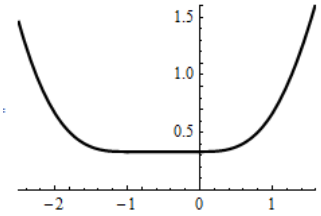


Grid[{{

$z = \int y[x]\, dx$,

Plot[z, {x, -2.5, 1.6}, **PlotRange** → {{-2.5, 1.6}, {0, 1.6}}]}}]

$$\begin{cases} -x - x^2 - \frac{x^3}{3} & x \le -1 \\ \frac{1}{3} & -1 < x \le 0 \\ \frac{1}{3} + \frac{x^3}{3} & 0 < x \le 1 \\ \frac{2}{3} - x + x^2 & \text{True} \end{cases}$$



Проверить правильность вычисления интегралов можно дифференцированием

$$\mathbf{i1} = \int \int \frac{1}{x^2 + 4}\, dx\, dx$$

$$\frac{1}{2}\left(x \text{ArcTan}[\frac{x}{2}] - \text{Log}[4 + x^2]\right)$$

FullSimplify[D[**i1**, {x, **2**}]]

$$\frac{1}{4 + x^2}$$

Напомним, что неопределенный интеграл от функции можно вычислить, используя функцию `Derivative`.

$$\mathbf{Derivative}[-1]\left[\frac{\#^2}{\#^3 + 1}\&\right][x]$$

$$\frac{1}{3}\text{Log}[1 + x^3]$$

Когда *Mathematica* не может взять интеграл, она возвращает команду обратно

$$\mathbf{Integrate}\left[\frac{1}{x + \mathbf{Cos}[x]}, x\right]$$

$$\int \frac{1}{x + \text{Cos}[x]}\, dx$$

Иногда для вычисления неопределенного интеграла следует учитывать условия, накладываемые на независимую переменную, поскольку без дополнительных условий интеграл не вычисляется. Например, условия важны при интегрировании кусочных функций

Integrate[**Abs**[x], x]

$\int \text{Abs}[x]\, dx$

Integrate[Abs[x], x, Assumptions−> x ∈ Reals]

$$\begin{cases} -\frac{x^2}{2} & x \le 0 \\ \frac{x^2}{2} & \text{True} \end{cases}$$

Заметим, что разные формы одного и того же неопределенного интеграла могут давать ответы, отличающиеся константой. Например

i1 = Integrate[1 + (x + 1)^3, x]

$$x + \frac{1}{4}(1+x)^4$$

i2 = Integrate[Expand[1 + (x + 1)^3], x]

$$2x + \frac{3x^2}{2} + x^3 + \frac{x^4}{4}$$

Simplify[i1 − i2]

$$\frac{1}{4}$$

Для вычисления определенного интеграла используется второй аргумент в виде списка `Integrate[f,{x,xmin,xmax}]`. Следующая команда интегрирует x^2 от 0 до 1.

Integrate[x^2, {x, 0, 1}]

$$\frac{1}{3}$$

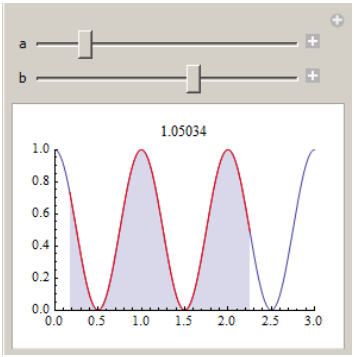
или

$$\int_0^1 x^2\, dx$$

Трафарет ввода определенного интеграла можно найти на панели специальных символов «Basic Math Input». Можно также использовать специальные комбинации клавиш. Для ввода «крючка» интеграла наберите `Esc-int-Esc`. Потом надо ввести подстрочный и надстрочный индексы у значка интеграла ∫. Для этого введите `Ctrl-_` (подчеркивание), затем `Ctrl-%` (или `Ctrl-^`, затем `Ctrl-%`). Введите значения нижнего и верхнего пределов интегрирования. `Ctrl-пробел` возвращает курсор на нормальный уровень ввода, где введите подынтегральное выражение, затем введите специальный значок дифференциала `Esc-dd-Esc` и имя переменной интегрирования.

Следующий код поясняет геометрический смысл определенного интеграла $\int_a^b f(x)dx$ как площади под кривой $y = f(x)$ на участке от a до b.

```
Manipulate[
  p1 = Plot[Cos[πx]^2, {x, 0, 3}, PlotRange → {{0, 3}, {0, 1}}];
  p2 = Plot[Cos[πx]^2, {x, a, b}, Filling → Axis, PlotStyle → Red,
                                  PlotRange → {{0, 3}, {0, 1}}];
  s = Integrate[Cos[πx]^2, {x, a, b}];
  Show[p1, p2, PlotLabel → s],
{{a, 0}, 0, 1}, {{b, 2}, 1.1, 3}]
```



Перемещая бегунки, связанные с параметрами a и b, вы можете наблюдать за областью, площадь которой вычисляется и отображается в заголовке.

Если под интегралом присутствует параметр, то результат будет от него зависеть и в ответе может присутствовать условие

$$\int_0^1 x^n \, dx$$

$$\text{ConditionalExpression}\left[\frac{1}{1+n}, \text{Re}[n] > -1\right]$$

Здесь `ConditionalExpression[expr,cond]` является символической конструкцией, которая представляет выражение `expr` только, если условие `cond` истинно. Вы можете использовать `ConditionalExpression` как функцию. Например,

```
y[n_] = Integrate[x^n, {x, 0, 1}];
y[5]
```

$$\frac{1}{6}$$

При вычислении определенного интеграла условие можно учитывать сразу. Для этого используется опция `Assumptions` или функция `Assuming`.

```
Integrate[x^n, {x, 0, 1}, Assumptions -> n > 0]
```

$$\frac{1}{1+n}$$

или

```
Assuming[n > 0, Integrate[x^n, {x, 0, 1}]]
```

$$\frac{1}{1+n}$$

У функции `Assuming` два аргумента: первый – список предположений/условий, второй – обрабатываемое выражение.

Можно отключить «условный» вывод результата с помощью опции `GenerateConditions → False`.

```
Integrate[x^n, {x, 0, 1}, GenerateConditions -> False]
```

$$\frac{1}{1+n}$$

Mathematica может вычислять несобственные интегралы с бесконечными пределами интегрирования

$$\int_0^\infty \frac{1}{1+x^4}\,dx$$

$$\frac{\pi}{2\sqrt{2}}$$

```
Integrate[Exp[-cx^2], {x, -∞, ∞}, Assumptions → Re[c] > 0]
```

$$\frac{\sqrt{\pi}}{\sqrt{c}}$$

При разных условиях/предположениях интеграл может принимать разные значения

```
Table[
      Integrate[(Cos[x](1 - Cos[ax]))/x^2, {x, 0, ∞}, Assumptions :→ as],
{as, {a > 1, -1 < a < 1, a < -1}}]
```

$$\left\{\frac{1}{2}(-1+a)\pi,\ 0,\ -\frac{1}{2}(1+a)\pi\right\}$$

Можно вычислять несобственные интегралы на конечных отрезках.

```
Integrate[1/√x, {x, 0, 1}]
```

2

Можно вычислять главное значение определенного интеграла, используя опцию `PrincipalValue->True`. Например, следующий интеграл не существует

```
Integrate[1/x, {x, -1, 3}]
```

```
Integrate::idiv: Integral of 1/x does not converge on {-1,3}.≫
```

$$\int_{-1}^{3} \frac{1}{x}\,dx$$

Но существует его главное значение.

```
Integrate[1/x, {x, -1, 3}, PrincipalValue-> True]
```

Log[3]

Аналогично у следующего интеграла существует только главное значение

```
Integrate[1/Sinh[2x + 1], {x, -1, 1}, PrincipalValue → True]
```

$$\text{ArcTanh}\left[\frac{e}{1+e^2}\right]$$

Вы можете пожелать строить графики неопределенных интегралов или определенных с переменными пределами интегрирования. Но при этом следует учитывать некоторые особенности. Например, следующая команда не работает

```
Plot[Integrate[Cos[x], x], {x, -4, 4}]
```

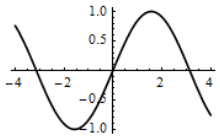
потому, что функция `Plot` имеет атрибут `HoldAll`.

```
Attributes[Plot]
```

{HoldAll, Protected, ReadProtected}

Он означает, что аргументы функции `Plot` не вычисляются символьно при выполнении команды. Чтобы вычислить выражения, предаваемые в качестве аргументов такой функции, следует использовать функцию `Evaluate`.

$$\mathbf{Plot}[\mathbf{Evaluate}[\mathbf{Integrate}[\mathbf{Cos}[x],x]],\{x,-4,4\}]$$



Следующая команда выполняется, но достаточно долго

$$\mathbf{Plot}[\mathbf{Integrate}[\mathbf{Cos}[t],\{t,-\pi,x\}],\{x,-\pi,\pi\}] \quad (* \text{ не делайте так } *)$$

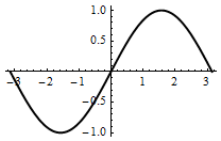
потому, что интеграл вычисляется символьно при каждом значении *x*. Чтобы интеграл вычислялся сразу (один раз) следует использовать функцию `Evaluate`.

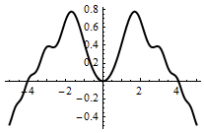
$$\mathbf{Plot}[\mathbf{Evaluate}[\mathbf{Integrate}[\mathbf{Cos}[t],\{t,-\pi,x\}]],\{x,-\pi,\pi\}] \quad (* \text{ след. рис. слева } *)$$

То же замечание касается такого примера (следующий рисунок справа).

$$\mathbf{Plot}[\mathbf{Evaluate}[\int_0^x \int_0^h (\mathbf{Cos}[ht]-0.1)\,dt\,dh],\{x,-5,5\},\mathbf{Axes}\to\mathbf{True}]$$

В этом примере без использования функции `Evaluate` на некоторых системах код выполняется так долго, что возможно вам придется прервать вычисления командой `Alt-.` (точка) или `Alt-,` (запятая).





Определенные интегралы, зависящие от параметра, можно дифференцировать по этому параметру. Например,

$$\mathbf{Integrate}[f[t],\{t,0,x\}]$$

$$\int_0^x f[t]\,dt$$

$$D[\%,x]$$

$$f[x]$$

Общая формула дифференцирования интеграла, зависящего от параметра, может быть получена следующей командой

$$\mathbf{Clear}[F,f,g]$$

$$\mathbf{Integrate}[F[t,x],\{t,g[x],f[x]\}]$$

$$\int_{g[x]}^{f[x]} F[t,x]\,dt$$

$$D[\%,x]$$

$$\int_{g[x]}^{f[x]} F^{(0,1)}[t,x]\,dt + F[f[x],x]f'[x] - F[g[x],x]g'[x]$$

В одной команде можно комбинировать вычисление определенного и неопределенного интегралов.

$$\mathbf{Integrate}[(x+y)^2,\{y,0,1\},x]$$

$$\frac{1}{6}x\,(2+3x+2x^2)$$

Для вычисления повторных интегралов добавьте больше аргументов в команду `Integrate`. Например, команда

```
Integrate[f,{x,xmin,xmax},{y,ymin,ymax}]
```

вычисляет повторный интеграл вида $\int_{x_{\min}}^{x_{\max}} d\,x \int_{y_{\min}}^{y_{\max}} f(x,y)\,d\,y$. Первым указывается интервал изменения той переменной, интегрирование по которой выполняется в последнюю очередь!

```
Integrate[x y,{x,0,π},{y,0,x}]
```

$$\frac{\pi^4}{8}$$

или

$$\int_0^{\pi}\int_0^{x} x\,y\,dy\,dx$$

$$\frac{\pi^4}{8}$$

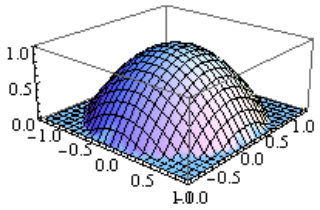
Функцию `Integrate` можно использовать для вычисления интегралов по области. Для этого можно использовать повторный интеграл предварительно «обнулив» функцию вне области. Вот пример интегрирования функции $f(x,y)=1-x^2-y^2$ по области единичного круга.

```
Integrate[If[x^2 + y^2 < 1, 1 - (x^2 + y^2), 0],{x,-1,1},{y,-1,1}]
```

$$\frac{\pi}{2}$$

Ниже показан график «обрезанной» функции $f(x,y)$.

```
Plot3D[If[x^2 + y^2 < 1, 1 - (x^2 + y^2), 0],{x,-1,1},{y,-1,1},
                                                    Mesh → {21,21}]
```



Тот же интеграл можно вычислить по – другому

```
Integrate[(1 - (x^2 + y^2))Boole[x^2 + y^2 < 1],{x,-1,1},{y,-1,1}]
```

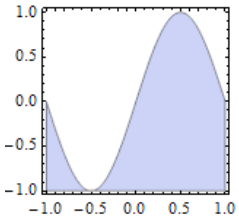
$$\frac{\pi}{2}$$

Функция `Boole[expr]` возвращает единицу, если `expr` принимает значение `True` и $0-$ если `False`.

Область может задаваться логической комбинацией неравенств

```
RegionPlot[-1 < x < 1 && y < Sin[πx]  && y > -1,
          {x,-1,1},{y,-1,1}, AspectRatio → Automatic]
Integrate[(x + y)Boole[-1 < x < 1 && y < Sin[πx] &&  y > -1],
          {x,-1,1},{y,-1,1}]
```

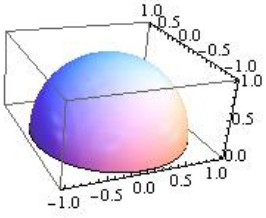
$$\frac{4-\pi}{2\pi}$$



Область может иметь любую размерность. В следующем примере мы вычисляем объем 3D области, ограниченной полусферой единичного радиуса.

```
Integrate[Boole[0 ≤ z ≤ 1 && x^2 + y^2 + z^2 ≤ 1],
    {x, -∞, ∞}, {y, -∞, ∞}, {z, -∞, ∞}]
RegionPlot3D[0 ≤ z ≤ 1 && x^2 + y^2 + z^2 ≤ 1,
    {x, -1, 1}, {y, -1, 1}, {z, 0, 1}, Mesh → None, BoxRatios → {1, 1, 0.5}]
```

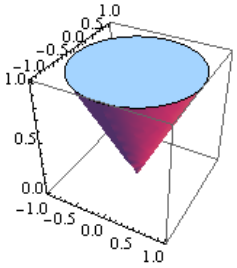
$$\frac{2\pi}{3}$$



Вот пример вычисления объема конуса.

```
Integrate[(x^2 + y^2)Boole[0 ≤ z ≤ 1 && x^2 + y^2 ≤ z^2],
        {x, -∞, ∞}, {y, -∞, ∞}, {z, -∞, ∞}]
RegionPlot3D[0 ≤ z ≤ 1 && x^2 + y^2 ≤ z^2,
        {x, -1, 1}, {y, -1, 1}, {z, 0, 1}, Mesh → None]
```

$$\frac{\pi}{10}$$

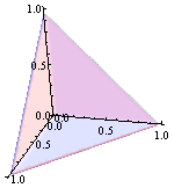


Пример. Вычислить интеграл

$$I = \iiint \frac{d\,x\,d\,y\,d\,z}{(1+x+y+z)^3}$$

распространенный на тетраэдр, ограниченный плоскостями $x=0,\ y=0,\ z=0$ и $x+y+z=1$. Вначале построим область интегрирования

```
RegionPlot3D[x > 0 && y > 0 && z > 0 && x + y + z ≤ 1,
        {x, 0, 1}, {y, 0, 1}, {z, 0, 1}, PlotPoints → 50, Mesh → None,
        PlotStyle → Opacity[0.5], Boxed → False, AxesOrigin → {0, 0, 0}]
```



Теперь вычисляем интеграл

```
Integrate[1/(1 + x + y + z)^3 Boole[x > 0 && y > 0 && z > 0 && x + y + z ≤ 1],
                                        {x, 0, 1}, {y, 0, 1}, {z, 0, 1}]
```

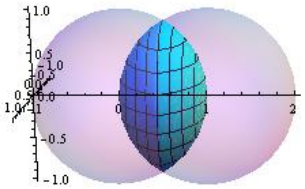
$$\frac{1}{16}(-5+8\text{Log}[2])$$

□

Пример. Найдем значение интеграла $I=\iiint y^2\,dx\,dy\,dz$ по области, являющейся общей частью двух шаров $x^2+y^2+z^2\le R^2$ и $x^2+y^2+z^2\le 2Ry$.

Вначале построим область интегрирования

```
R = 1;
opt = {PlotPoints → 50, PlotStyle → Opacity[0.3], Mesh → None,
       PlotRange → All};
sp1 = RegionPlot3D[x^2 + y^2 + z^2 ≤ R^2,
       {x, -R, R}, {y, -R, R}, {z, -R, R}, ##]& @@ opt;
sp2 = RegionPlot3D[x^2 + y^2 + z^2 ≤ 2Ry,
       {x, -R, R}, {y, 0, 2R}, {z, -R, R}, Evaluate[opt]];
sp3 = RegionPlot3D[x^2 + y^2 + z^2 ≤ R^2 && x^2 + y^2 + z^2 ≤ 2 R y,
       {x, -R, R}, {y, 0, R}, {z, -R, R}, PlotPoints → 50, PlotStyle → Cyan,
       MeshFunctions → {#2&, #3&}, Mesh → {7, 7, 7}];
Show[sp1, sp2, sp3, AxesOrigin → {0, -R, 0}, Boxed → False,
       BoxRatios → {2R, 3R, 2R}]
```



Обратите внимание на два способа применения списка опций **opt**. Теперь вычисляем интеграл

```
Integrate[y^2 Boole[x^2 + y^2 + z^2 ≤ R^2 && x^2 + y^2 + z^2 ≤ 2Ry],
       {x, -∞, ∞}, {y, -∞, ∞}, {z, -∞, ∞}, Assumptions :→ R > 0]
```

$$\frac{59\pi}{480}$$

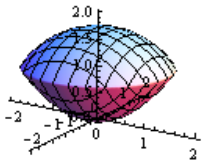
□

Пример. Вычислим интеграл $I=\iiint\limits_V (x+y+z)^2\,dx\,dy\,dz$ по области V, являющейся общей частью параболоида $x^2+y^2\le 2az$ и шара $x^2+y^2+z^2\le 3a^2$.

Вначале представим область интегрирования

```
a = 1;
```

```
RegionPlot3D[x^2 + y^2 ≤ 2 a z && x^2 + y^2 + z^2 ≤ 3a^2,
    {x, -2, 2}, {y, -2, 2}, {z, 0, 2}, PlotPoints → 100, Boxed → False,
    AxesOrigin → {0, 0, 0}, Mesh → {9, 9, 0}, BoxRatios → {4, 4, 2}]
```



Теперь вычисляем интеграл.

```
Integrate[(x + y + z)^2 Boole[x^2 + y^2 ≤ 2 a z && x^2 + y^2 + z^2 ≤ 3a^2],
                {x, -∞, ∞}, {y, -∞, ∞}, {z, -∞, ∞}]
```

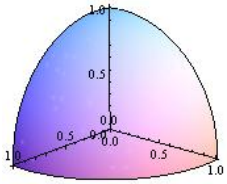
$$\frac{1}{30}(-97 + 108\sqrt{3})\pi$$

Пример. Вычислим интеграл

$$\iiint\limits_{\substack{x,y,z\geq 0 \\ x^2+y^2+z^2\leq R^2}} \frac{x\,y\,z\,d\,x\,d\,y\,d\,z}{\sqrt{\alpha^2x^2+\beta^2y^2+\gamma^2z^2}} \quad (\alpha>\beta>\gamma>0)$$

Вначале представим область интегрирования

```
R = 1;
RegionPlot3D[x ≥ 0&&y ≥ 0&&z ≥ 0&&x^2 + y^2 + z^2 ≤ R^2,
{x, 0, 1}, {y, 0, 1}, {z, 0, 1}, PlotPoints → 50, Mesh → None,
AxesOrigin → {0, 0, 0}, Boxed → False]
```



Теперь вычисляем интеграл.

```
Clear[a, b, c, R];
Block[{$Assumptions = a > b && b > c && c > 0 && R > 0},
  Integrate[xyz / Sqrt[a^2 x^2 + b^2 y^2 + c^2 z^2] *
            Boole[x ≥ 0 && y ≥ 0 && z ≥ 0 && x^2 + y^2 + z^2 ≤ R^2],
  {x, -∞, ∞}, {y, -∞, ∞}, {z, -∞, ∞}]]
```

$$\frac{(ab+ac+bc)R^5}{15(a+b)(a+c)(b+c)}$$

Обратите внимание на способ наложения условий на параметры. Вместо использования опции `Assumptions` мы использовали переменную `$Assumptions`, область действия которой ограничили блоком.

□

Для вычисления объема $4-$х мерного шара единичного радиуса можно выполнить команду

```
Integrate[Boole[Sum[x[i]^2, {i, 4}] < 1],
            {x[1], -1, 1}, {x[2], -1, 1}, {x[3], -1, 1}, {x[4], -1, 1}]
```

$$\frac{\pi^2}{2}$$

При вычислении интегралов по области иногда также следует указывать условия/предположения, накладываемые на параметры. Например, при вычислении объема n – мерного шара радиуса r следует использовать условие/предположение относительно параметра $\mathbf{r} > 0$.

```
n = 5;  (* размерность пространства *)
Integrate[Boole[Sum[x[i]^2, {i, n}] < r^2],
    Apply[Sequence, Table[{x[i], -∞, ∞}, {i, n}]], Assumptions → r > 0]
```

$$\frac{8\pi^2 r^5}{15}$$

Проверьте формулу для `n=2` и `n=3`.

Функцию `Integrate` можно применять к спискам выражений. В результате будет получен список интегралов.

```
Integrate[{x, x^2, x^3, x^4}, x]
```

$$\left\{\frac{x^2}{2}, \frac{x^3}{3}, \frac{x^4}{4}, \frac{x^5}{5}\right\}$$

```
Integrate[{x, x^2, x^3, x^4}, {x, 0, 1}]
```

$$\left\{\frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}\right\}$$

```
Integrate[{{1, 2x, 3x^2}, {2x, 1, 4x^3}, {3x^2, 4x^3, 1}}, x]//MatrixForm
```

$$\begin{pmatrix} x & x^2 & x^3 \\ x^2 & x & x^4 \\ x^3 & x^4 & x \end{pmatrix}$$

Функция `Integrate` умеет вычислять интегралы от комплексных функций вещественного аргумента. В частности для функций вида $f(t) = u(t) + \mathbf{i}\, v(t)$ имеем $\int f(t)\,dt = \int u(t)\,dt + \mathbf{i}\int v(t)\,dt$. При этом нет необходимости в выделении вещественной и мнимой частей функции. Например,

```
f[t_] = (t - i)^3;
u[t_] = ComplexExpand[Re[f[t]]];
v[t_] = ComplexExpand[Im[f[t]]]
i1 = ∫ f[t] dt
i2 = ∫ u[t] dt + i ∫ v[t] dt
Simplify[i1 - i2]
```

$$\frac{1}{4}(-i + t)^4$$

$$-\frac{3t^2}{2} + \frac{t^4}{4} + i(t - t^3)$$

```
1/4
```

Функция ComplexExpand[*expr*] выполняет разложение выражения `expr` в предположении, что все переменные вещественные. Как можно было ожидать, первообразные, полученные разными путями, отличаются только константой.

Аналогично можно вычислять определенные интегралы.

$$\int_0^2 f[t]\,dt$$

$-2-6i$

или

$$\int_0^2 u[t]\,dt + I\int_0^2 v[t]\,dt$$

$-2-6i$

Оба способа приводят к одинаковому результату.

Функция `Integrate` умеет вычислять интегралы по ломаным в комплексной плоскости. Для этого используется форма Integrate[$f[z],\{z,z_1,z_2,\ldots,z_k\}$], где z_i представляют комплексные числа (точки комплексной плоскости) и интегрирование выполняется по отрезкам от точки z_i до точки z_{i+1}. Например,

$$\mathbf{Integrate}\left[\frac{1}{z},\{z,1+I,-1+I,-1-I,1-I,1+I\}\right]$$

$2i\pi$

Тот же результат можно получить, суммируя интегралы по отрезкам.

$$\mathbf{mi}=\{\mathbf{Integrate}\left[\frac{1}{z},\{z,1+I,-1+I\}\right],\mathbf{Integrate}\left[\frac{1}{z},\{z,-1+I,-1-I\}\right],$$
$$\mathbf{Integrate}\left[\frac{1}{z},\{z,-1-I,1-I\}\right],\mathbf{Integrate}\left[\frac{1}{z},\{z,1-I,1+I\}\right]\}$$

$$\left\{\frac{i\pi}{2},\frac{i\pi}{2},\frac{i\pi}{2},\frac{i\pi}{2}\right\}$$

$$\mathbf{Total}[\mathbf{mi}]$$

$2\,i\,\pi$

Функция `Total[list]` возвращает сумму элементов списка `list`.

Вот еще пример интегрирования функции по ломаной в комплексной плоскости.

$$\mathbf{Integrate}\left[\frac{1}{z+1/2},\{z,1,e^{\frac{i\pi}{3}},e^{\frac{2i\pi}{3}},-1,e^{-\frac{2i\pi}{3}},e^{-\frac{i\pi}{3}},1\}\right]//\mathbf{FullSimplify}$$

$2i\pi$

Можно нарисовать контур интегрирования, например, так

$$\mathbf{pp}=\{\mathbf{Re}[\#],\mathbf{Im}[\#]\}\&/@\mathbf{Exp}\left[\frac{2\pi i}{6}\mathbf{Range}[0,6]\right]$$

$$\{\{1,0\},\{\frac{1}{2},\frac{\sqrt{3}}{2}\},\{-\frac{1}{2},\frac{\sqrt{3}}{2}\},\{-1,0\},\{-\frac{1}{2},-\frac{\sqrt{3}}{2}\},\{\frac{1}{2},-\frac{\sqrt{3}}{2}\},\{1,0\}\}$$

$$\mathbf{Graphics@Line@(pp)}$$

Функцию `Integrate` можно использовать для вычисления интегралов по кривым в комплексной плоскости. Если функция $f(z)$ определена на контуре C комплексной плоскости, который имеет параметрическое уравнение $z = z(t), a \le t \le b$, то имеет место формула

$$\int_C f(z)\mathrm{d}z = \int_a^b f(z(t))z'(t)\,dt$$

Вычислим интеграл функции $f(z) = \dfrac{1}{z+1/2}$ по единичной окружности. Ее параметрическое уравнение имеет вид $z(t) = e^{it}$, $0 \le t \le 2\pi$. Тогда

```
f[z_] = 1/(z + 1/2);
z[t_] = Exp[I t];
F[t_] = f[z[t]] z'[t]
```

$$\frac{i e^{it}}{\frac{1}{2} + e^{it}}$$

```
Integrate[F[t], {t, 0, 2π}]
```

$2\,i\,\pi$

3.3.2 Численное интегрирование

Когда *Mathematica* не может взять определенный интеграл, она возвращает команду обратно точно так же, как и при вычислении неопределенного интеграла

```
∫_0^1 Cos[10 Cos[x]] dx
```

$$\int_0^1 \mathrm{Cos}[10\,\mathrm{Cos}[x]]\,dx$$

Вы можете использовать функцию N для численного интегрирования

```
N[∫_0^1 Cos[10 Cos[x]] dx]
```

-0.301928

или

```
N[∫_0^1 Cos[10 Cos[x]] dx, 20]
```

-0.30192779721155889037

Если вы знаете, что хотите интегрировать численно, то вы можете использовать функцию `NIntegrate`. Это не разрешает *Mathematica* пробовать вычислять интеграл в символьном виде

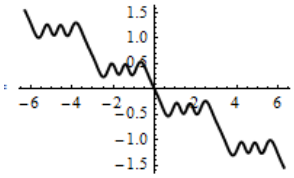
NIntegrate[Cos[10 Cos[x]], {x, 0, 1}]

-0.301928

Вы можете подумать, что неопределенный интеграл не может быть получен численно, потому что ответом является функция, а не число. Но это не так. Построим, например, график функции $F(x)=\int_0^x \cos(10\cos t)\,dt$, являющейся первообразной функции $\cos(10\cos x)$.

F[x_] := NIntegrate[Cos[10 Cos[t]], {t, 0, x}];

Plot[F[x], {x, −2π, 2π}]



Важным моментом в приведенном коде является отложенное присваивание при создании функции F[x]. Функция `Plot` при построении графика сначала выбирает значение переменной *x* из заданного интервала и подставляет ее в аргумент функции F[x]. Поскольку присваивание отложенное, то вычисление интеграла происходит численно при этом значении *x* и получается точка графика. Затем переменной *x* функция `Plot` назначает другое значение и вычисление интеграла происходит при этом новом значении. Каждый раз интеграл вычисляется численно при новом значении верхнего предела *x*. Затем из полученного множества точек функция `Plot` строит график.

Для построения графика первообразной можно поступить по – другому. Аналогично тому, как в *Mathematica* приближаются числа, вы можете приблизить функцию F(*x*) другой функцией. Для этого нужно вычислить значение F(*x*) в некотором наборе точек и построить интерполяционную функцию. Эту последовательность действий выполняет функция `NDSolve`, которая в общем случае численно решает дифференциальное уравнение. В следующем примере мы построим график первообразной функции $\cos(10\cos x)$ в интервале от -10 до 10.

ss = NDSolve[{y'[x] == Cos[10 Cos[x]], y[0] == 0}, y[x], {x, −2π, 2π}]

{{y[x] → InterpolatingFunction[{{−6.28319,6.28319}}, " <> "][x]}}

Plot[y[x] /. ss, {x, −2π, 2π}] (* предыдущий график *)

Здесь функция `NDSolve` построила приближенное решение дифференциального уравнения $y'(x)=\cos(10\cos x)$ с начальным условием $y(0)=0$ в виде интерполяционной функции `InterpolatingFunction`.

О функции `NDSolve` мы будем говорить подробно в следующей главе. Вы можете использовать последний пример в качестве шаблона. Замените Cos[10 Cos[x]] вашей функцией, а интервал от -10 до 10 на такой, где вы хотите получить результат.

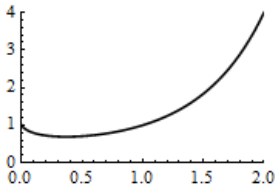
Основное назначение функции `NIntegrate` состоит в приближенном вычислении определенных интегралов – однократных, повторных или (много)кратных.

Ее можно использовать для вычисления интегралов по конечному отрезку вещественной оси.

$$\mathbf{NIntegrate}[x^x, \{x, 0, 2\}]$$

$$\mathbf{Plot}[x\text{^}x, \{x, 0, 2\}, \mathbf{PlotRange} \to \{\{0, 2\}, \{0, 4\}\}]$$

```
2.83388
```



Можно численно находить интеграл на бесконечном участке вещественной оси

$$\mathbf{NIntegrate}[\frac{1}{1+x^2}, \{x, 0, \infty\}]$$

```
1.5708
```

Численное интегрирование можно выполнять по конечному или бесконечному отрезку в комплексной плоскости.

$$\mathbf{NIntegrate}[\sqrt{x}, \{x, I, 3 - I\}]$$

$$3.79214 - 2.21131i$$

Обычно второй аргумент функции `NIntegrate` имеет вид короткого списка $\{x, x_{min}, x_{max}\}$. Но для подынтегральных выражений с особенностями его можно задавать в виде $\{x, x_0, x_1, \ldots, x_n\}$. В этом случае выполняется проверка наличия особенности в каждой из внутренних точек x_i. Если особенностей нет, то результат эквивалентен интегралу от x_0 до x_n.

$$\mathbf{NIntegrate}[1/\mathbf{Sqrt}[x], \{x, -1, 1\}]$$

```
Numerical integration converging too slowly...
```

$$\mathbf{NIntegrate}[1/\mathbf{Sqrt}[x], \{x, -1, 0, 1\}]$$

```
2. -2.i
```

Числа $x, x_0, x_1, \ldots, x_n$ могут быть комплексными и тогда интеграл будет представлять интеграл по ломаной в комплексной плоскости.

$$f[z\_] = \frac{1 - e^z + z}{z^3(z-1)^2};$$

$$\mathbf{Chop}[\mathbf{NIntegrate}[f[z], \{z, 2 - 2I, 3 + 2I, -3 + I, -1 - I, 2 - 2I\}]]$$

$$0. - 0.398582\, i$$

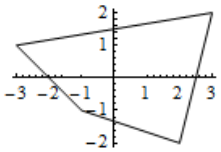
Функция `Chop[expr]` заменяет нулем в выражении `expr` все близкие к нулю числа, поскольку обычно они являются погрешностью вычислений.

Контур, по которому выполнено интегрирование, показан на следующем рисунке.

$$\mathbf{pp} = \{\mathbf{Re}[\#], \mathbf{Im}[\#]\}\&/@\{2 - 2I, 3 + 2I, -3 + I, -1 - I, 2 - 2I\}$$

$$\mathbf{Show}[\mathbf{Graphics@Line@pp}, \mathbf{Axes} \to \mathbf{True}]$$

$$\{2, -2\}, \{3,2\}, \{-3,1\}, \{-1, -1\}, \{2, -2\}\}$$



Можно численно интегрировать по контуру на комплексной плоскости. В следующем примере вычисляется интеграл $\oint_C \frac{1}{z+1} dz$, где C – окружность радиуса 2 с центром в начале координат.

```
Chop[NIntegrate[(i 2 Exp[i t])/(2 Exp[i t] + 1), {t, 0, 2π}]]
```

```
0.+6.2831853 i
```

Можно интегрировать списки выражений

```
NIntegrate[{x, x^2, x^3}, {x, 0, 1}]
```

```
{0.5,0.33333333,0.25}
```

Функция `NIntegrate` умеет вычислять повторные интегралы.

```
NIntegrate[Exp[-x^2 + y], {x, 0, ∞}, {y, 0, 1}]
```

```
1.5227876
```

Внутренний список $\{x, 0, \infty\}$ определяет внешний интеграл. Последним указывается интервал изменения той переменной, интегрирование по которой выполняется в первую очередь. При этом пределы интегрирования могут зависеть от переменной внешнего интеграла. Например, площадь единичного круга можно вычислить с помощью следующего интеграла

```
NIntegrate[1, {x, -1, 1}, {y, -Sqrt[1 - x^2], Sqrt[1 - x^2]}]
```

```
3.1415927
```

В следующем примере мы вычисляем трехкратный интеграл с особенностью в начале координат

```
NIntegrate[1/Sqrt[x^2 + y^2 + z^2], {x, 0, 1}, {y, 0, 1}, {z, 0, 1}]
```

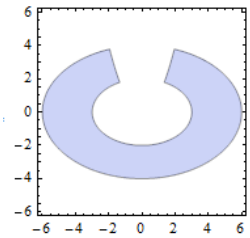
```
1.19004
```

Можно выполнять численное интегрирование по области. В следующем примере мы вычисляем площадь области эллиптического кольца с параболическим вырезом.

```
NIntegrate[Boole[1 ≤ x^2/9 + y^2/4 ≤ 4 && y < x^2], {x, -6, 6}, {y, -6, 6}]
```

```
RegionPlot[1 ≤ x^2/9 + y^2/4 ≤ 4 && y < x^2, {x, -6, 6}, {y, -6, 6}]
```

```
49.7429
```



У функции NIntegrate имеются опции, которые управляют точностью вычислений.

AccuracyGoal определяет абсолютную погрешность вычислений, используя количество значащих цифр. Значение по умолчанию AccuracyGoal → ∞ говорит, что эта опция не должна использоваться как критерий прекращения вычислений (т.е. будут использоваться другие опции).

Когда значение интеграла равно нулю функция NIntegrate возвращает сообщение о медленной сходимости алгоритма. В такой ситуации требуется установка опции AccuracyGoal в значение, отличное от установленного по умолчанию (от Infinity) и критерием остановки вычислений будет абсолютная погрешность.

NIntegrate[Sin[x], {x, 0, 2π}]

```
Numerical integration converging too slowly; suspect one of the
following: singularity, value of the integration is 0, ...
```

Но

NIntegrate[Sin[x], {x, 0, 2π}, AccuracyGoal → 6]

-2.886686×10^{-16}

PrecisionGoal определяет «относительную погрешность» вычислений и задается количеством значащих цифр; PrecisionGoal → ∞ указывает, что эта опция не будет использоваться как критерий прекращения вычислений (будет использоваться опция AccuracyGoal). Для опции PrecisionGoal мы используем не совсем корректный термин «относительная погрешность». PrecisionGoal->n определяет для значения *x* погрешность вычислений равную $|x|10^{-n}$. Когда включены опции AccuracyGoal->m и PrecisionGoal->n, то *Mathematica* пытается выполнять вычисления величины *x* с погрешностью не более $10^{-m}+|x|10^{-n}$.

Опция WorkingPrecision определяет количество значащих цифр, используемых во внутренних вычислениях; значение этой опции, как правило, должно быть больше значения опции AccuracyGoal; установка WorkingPrecision->MachinePrecision приводит к вычислениям с процессорной точностью.

NIntegrate[$\frac{1}{1+\sqrt{x}}$, {x, 0, 1}, PrecisionGoal → 20, WorkingPrecision → 40]

0.61370563888010938116553575708364686389 90

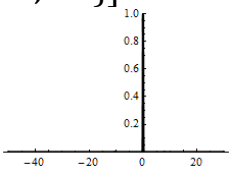
Функция NIntegrate выполняет разбиение участка интегрирования до тех пор, пока ошибка вычислений выбранного ею метода не станет меньше погрешности, определяемой опциями AccuracyGoal и PrecisionGoal. Функция NIntegrate использует адаптивные алгоритмы с автоматическим выбором шага интегрирования. Эти алгоритмы сами, если это требуется, рекурсивно дробят отрезки разбиения области интегрирования в местах быстрого изменения функции. Опция MinRecursion, определяет количество шагов дробления отрезков разбиения, с которого начинаются

вычисления. Опция MaxRecursion определяет максимальное количество шагов дробления.

Рассмотрим пример. Функции NIntegrate может пропустить участки быстрого возрастания подынтегрального выражения. Вот пример такой функции.

$$\mathbf{Plot}[\mathbf{Exp}[-\mathbf{100}x^{\mathbf{2}}],\{x,-\mathbf{50},\mathbf{30}\},\mathbf{PlotRange}\to\{\mathbf{0},\mathbf{1}\},\mathbf{PlotPoints}\to\mathbf{1000}]$$

$$\mathbf{NIntegrate}[\mathbf{Exp}[-\mathbf{100}x^{\mathbf{2}}],\{x,-\mathbf{50},\mathbf{30}\}]$$




```
NIntegrate failed to converge to prescribed accuracy after 9
recursive bisections ...
```

Увеличив значение опция MinRecursion, можно вынудить функцию NIntegrate в самом начале делать более мелкое дробление участков разбиения.

$$\mathbf{NIntegrate}[\mathbf{Exp}[-\mathbf{100}x^{\mathbf{2}}],\{x,-\mathbf{50},\mathbf{30}\},\mathbf{MinRecursion}\to\mathbf{4}]$$

```
0.177245
```

По умолчанию установлено MinRecursion->0. Опция MaxRecursion→n определяет, что рекурсивно можно дробить отрезки разбиения n раз. Рекурсивное/многократное деление выполняется только на тех участках, где значения функции быстро изменяются и для достижения требуемой точности длина отрезков разбиения должна быть уменьшена.

Опция MaxPoints ограничивает количество точек разбиения интервала интегрирования.

Опция Exclusions позволяет исключить из рассмотрения некоторые особые точки (кривые или поверхности для кратных интегралов) из области интегрирования.

$$\mathbf{NIntegrate}[\mathbf{Log}[x-\mathbf{2}],\{x,\mathbf{0},\mathbf{4}\}]$$

```
Numerical integration converging too slowly; suspect one of the
following: singularity,...
```

$$\mathbf{NIntegrate}[\mathbf{Log}[x-\mathbf{2}],\{x,\mathbf{0},\mathbf{4}\},\mathbf{Exclusions}\to\{\mathbf{2}\}]$$

```
-1.2274113+6.2831853 i
```

В функции NIntegrate реализовано много различных алгоритмов численного интегрирования. Обычно NIntegrate сама выбирает метод. Однако иногда вы будете задавать метод с помощью опции Method. Мы не ставим перед собой задачу изложения всех методов, используемых этой функцией. С ними вы можете познакомиться по справочной системе.

В частности, если вы хотите численно найти главное значение интеграла, то следует использовать опцию Method→"PrincipalValue".

$$\mathbf{NIntegrate}[\frac{\mathbf{1}}{x},\{x,-\mathbf{1},\mathbf{2}\},\mathbf{Method}\to\text{"}\mathbf{PrincipalValue}\text{"},\ \mathbf{Exclusions}\to x==\mathbf{0},\mathbf{AccuracyGoal}\to\mathbf{8}]$$

```
0.69314718
```

Попробуйте удалить хотя бы одну из опций предыдущего примера и вы не поучите результат или получите предупреждение системы о медленной сходимости.

Опцией `EvaluationMonitor` задается выражение, которое вычисляется каждый раз, когда вычисляется подынтегрального выражение. В следующем примере определяется количество точек, использованных при численном интегрировании

$$\mathbf{Block}[\{k = 0\}, \{\mathbf{NIntegrate}[x^3, \{x, 0, 2\}, \mathbf{EvaluationMonitor} :\to k++], k\}]$$

```
{4.,11}
```

Функция `Block[{x=x0,y=y0,...},expr]` выполняет блок кода с локальными переменными `x,y,...`, для которых заданы начальные значения x_0, y_0 и т.д.

Значение интеграла равно 4 и использовано всего 11 точек. Можно даже узнать эти точки.

$$\mathbf{Block}[\{X = \{\}\}, \{\mathbf{NIntegrate}[x^3, \{x, 0, 2\}, \mathbf{EvaluationMonitor} :\to (X = \mathbf{Join}[X, \{x\}])], X\}]$$

```
{4.,
{0.01591464,0.093820154,0.24583327,0.46153069,0.72036959,
1.,1.2796304,1.5384693,1.7541667,1.9061798,1.9840854}}
```

Те же действия можно выполнить командой

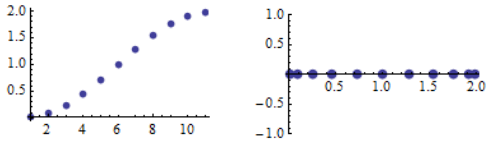
$$r = \mathbf{Reap}[\mathbf{NIntegrate}[x^3, \{x, 0, 2\}, \mathbf{EvaluationMonitor} :> \mathbf{Sow}[x]]]$$

```
{4.,
{{0.01591464,0.093820154,0.24583327,0.46153069,0.72036959
,1.,1.2796304,1.5384693,1.7541667,1.9061798,1.9840854}}}
```

Функция `Sow[`*выражение*`]` «сеет» значения *выражения*, которые «собираются» ближайшей охватывающей функцией `Reap[expr]`, которая также возвращает значение выражения `expr`.

Можно построить график значений этих точек (следующий рисунок слева)

$$\mathbf{ListPlot}[r[[2, 1]], \mathbf{PlotRange} -> \mathbf{All}, \mathbf{PlotStyle} -> PointSize[0.02]]$$



или показать, как эти точки расположены на отрезке интегрирования (предыдущий рисунок справа)

$$\mathbf{pp} = \mathbf{Transpose}[\{r[[2, 1]], \mathbf{ConstantArray}[0, \mathbf{Length}[r[[2, 1]]]]\}]$$

```
{{0.01591464,0},{0.093820154,0},{0.24583327,0},
 {0.46153069,0},{0.72036959,0},...,{1.9840854,0}}
```

$$\mathbf{ListPlot}[\mathbf{pp}, \mathbf{PlotStyle} -> \mathbf{PointSize}[0.03]]$$

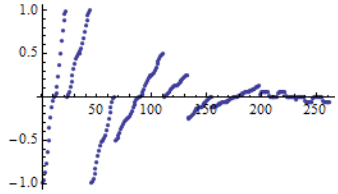
Для несобственного интеграла точек разбиения много больше

$$\mathbf{Block}[\{k = 0\}, \{\mathbf{NIntegrate}[\frac{1}{\sqrt{x}}, \{x, 0, 1\}, \mathbf{EvaluationMonitor} :\to k++], k\}]$$

```
{2.,131}
```

Следующий график поясняет как функция `NIntegrate` размещает точки разбиения

```
rp = Reap[NIntegrate[1/Sqrt[x], {x, -1, 0, 1},
                     EvaluationMonitor:> Sow[x]]];
ListPlot[rp[[2, 1]], PlotRange-> All]
```



По горизонтали отложены номера точек, а по вертикали – координаты точек на оси X. По графику можно проанализировать стратегию выбранного функцией `NIntegrate` метода. Первые ≈ 20 точек размещаются более менее равномерно на отрезке [-1, 1]. Следующие ≈ 50 точек размещаются на отрезках интегрирования, образуемых первыми точками. На следующем шаге функция `NIntegrate` разбивает отрезки из интервала (-0.5, 0.5). Потом разбиваются отрезки из интервала (-0.25, 0.25). Далее функция `NIntegrate` выполняет еще несколько шагов, дробя отрезки интегрирования, расположенные в окрестности нуля.

3.3.3 Вычисление длин, площадей и объемов

Длина кривой.

Длина дуги плоской кривой, заданной явным уравнением $y = y(x)$ в декартовых координатах, вычисляется по формуле

$$L = \int_{x_0}^{x_1} \sqrt{1 + \left(y'(x)\right)^2}\, dx, \qquad (1)$$

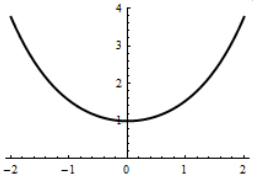
где x_0 и x_1 определяют начальную и конечную точки дуги.

Пример. Длина цепной линии $y = a\,ch\frac{x}{a}$.

```
Clear[a, x, y, t];
y[x_] = aCosh[x/a];
L = Integrate[Sqrt[1 + D[y[t], t]^2], {t, 0, x}, Assumptions :→ {x ∈ Reals, a > 0}]
```

$$a\,\mathrm{Sinh}\left[\frac{x}{a}\right]$$

```
a = 1; Plot[y[x], {x, -2, 2}, PlotRange → {0, 4}]
```



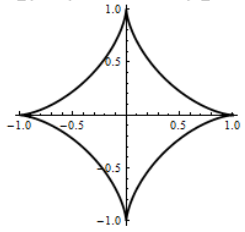
Длина дуги плоской кривой, заданной в параметрическом виде $x = x(t),\ y = y(t)$, вычисляется по формуле

$$L = \int_{t_0}^{t_1} \sqrt{(x'(t))^2 + (y'(t))^2}\, dt \qquad (2)$$

где t_0 и t_1 – значения параметра t в начальной и конечной точках дуги.

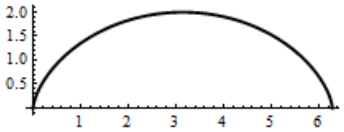
Пример. Длина астроиды $x = a\cos^3 t$, $y = a\sin^3 t$, $0 \le t \le 2\pi$.

```
Clear[a, x, y]
x[t_] = a Cos[t]^3;
y[t_] = a Sin[t]^3;
L = Integrate[Sqrt[D[x[t], t]^2 + D[y[t], t]^2], {t, 0, 2 π}, Assumptions :→ a > 0]
6 a
a = 1; ParametricPlot[{x[t], y[t]}, {t, 0, 2 π}]
```



Пример. Длина циклоиды $x = a(t - \sin t)$, $y = a(1 - \cos t)$, $0 \le t \le 2\pi$.

```
Clear[a]
x[t_] = a(t - Sin[t]);
y[t_] = a(1 - Cos[t]);
L = Integrate[Sqrt[D[x[t], t]^2 + D[y[t], t]^2], {t, 0, 2 π}, Assumptions :→ a > 0]
8 a
a = 1; ParametricPlot[{x[t], y[t]}, {t, 0, 2 π}]
```



Пример. Длина дуги эллипса.

```
Clear[a, b, x, y, t]
x[t_] = a Sin[t];
y[t_] = b Cos[t];
```

В следующей строке кода мы выполняем преобразование подынтегрального выражения $\sqrt{(x'(t))^2 + (y'(t))^2}$ к виду, с которым может «справиться» функция `Integrate` (вычитаем и добавляем a^2, делим на a^2 под корнем, умножаем корень на a, множитель $1 - b^2/a^2$ обозначаем p^2).

```
dl = a Simplify[Sqrt[(Simplify[D[x[t], t]^2 + D[y[t], t]^2 - a^2] + a^2)/a^2]] /. {(-1 + b^2/a^2) → -p^2}
a Sqrt[1 - p^2 Sin[t]^2]
```

Величина $p=\sqrt{1-b^2/a^2}$ называется эксцентриситетом эллипса, если b меньшая его полуось. Вычисляя длину дуги эллипса от верхнего конца малой оси до любой его точки в первом квадранте, получим

$$\mathbf{Integrate[dl,\{t,0,\theta\},Assumptions:\to\{0<p<1,0<\theta<\pi\}]/.p^2\to 1-\frac{b^2}{a^2}}$$

$$a\ \text{EllipticE}\left[\theta,1-\frac{b^2}{a^2}\right]$$

Здесь мы вернулись к прежним обозначениям. Т.о. длина дуги эллипса выражается эллиптическим интегралом 2-го рода; этот факт послужил поводом для самого названия «эллиптический».

Длина всего обвода эллипса равна

$$\mathbf{L=Integrate[dl,\{t,0,2\pi\},Assumptions:\to\{0<p<1\}]/.p^2\to 1-\frac{b^2}{a^2}}$$

$$4\ a\ \text{EllipticE}\left[1-\frac{b^2}{a^2}\right]\quad \text{(* полный эллиптический интеграл *)}$$

Численное значение длины эллипса получается использованием функции N.

```
b = 1; a = 2; N[L]
9.6884482
```

Длина дуги кривой, заданной в полярных координатах уравнением $r=r(\varphi)$, вычисляется по формуле

$$L=\int_{\varphi_0}^{\varphi_1}\sqrt{r^2(\varphi)+(r'(\varphi))^2}\,d\varphi \qquad (3)$$

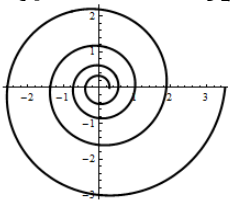
где φ_0 и φ_1 – значения углового параметра φ в начальной и конечной точках дуги.

Пример. Длина логарифмической спирали $r=a\,e^{m\varphi}$.

```
Clear[a, m]
r[φ_] = a Exp[m φ];
L = Integrate[√(r[φ]^2 + D[r[φ], φ]^2), {φ, 0, θ}, Assumptions :→ {a > 0, m > 0}]
```

$$\frac{a(-1+e^{m\theta})\sqrt{1+m^2}}{m}$$

```
a = 1; m = 0.1; PolarPlot[r[φ], {φ, -4π, 4π}]
```



Длина дуги пространственной кривой, заданной параметрически $x=x(t),\ y=y(t),\ z=z(t)$, вычисляется по формуле

$$L=\int_{t_0}^{t_1}\sqrt{(x'(t))^2+(y'(t))^2+(z'(t))^2}\,dt \qquad (4)$$

где t_0 и t_1 – значения параметра t в начальной и конечной точках дуги.

Пример. Длина винтовой линии $x = a\cos t$, $y = a\sin t$, $z = ct$ от точки $t = 0$ до точки $t = T$ будет равна

```
Clear[a, c, x, y, z]
{x[t_], y[t_], z[t_]} = {a Cos[t], a Sin[t], ct};
L = Integrate[√(D[x[t], t]^2 + D[y[t], t]^2 + D[z[t], t]^2), {t, 0, T}]
```

$\sqrt{a^2 + c^2}T$

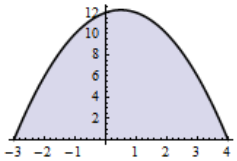
Площадь плоской фигуры.

Геометрический смысл определенного интеграла $\int_a^b f(x)dx$ состоит в том, что он представляет *площадь под кривой* $y = f(x)$ при $a \le x \le b$.

Пример. Найдем площадь фигуры, ограниченной сверху параболой $y = 9 - x^2$, а снизу осью абсцисс. Для этого решим уравнение $12 + x - x^2 = 0$ и найдем точки, в которых парабола пересекается с осью абсцисс.

```
y[x_] = -x^2 + x + 12;
{x1, x2} = x /. Solve[y[x] == 0, x]
Plot[y[x], {x, x1, x2}, Filling → Axis]
```

{-3,4}

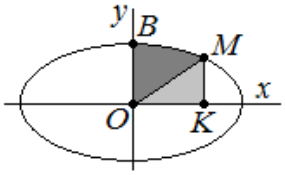


Теперь вычисляем площадь

```
S = Integrate[y[x], {x, x1, x2}]
```

$\frac{343}{6}$

Пример. Даны эллипс $\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$ и точка $M(x, y)$ на нем. Определить площадь криволинейной трапеции BOKM и сектора OMB.



Из уравнения эллипса имеем

```
y[x_] = b/a √(a^2 - x^2);
SBOKM = Apart[Integrate[y[t], {t, 0, x},
                        Assumptions :→ {a > 0, b > 0, 0 < x < a}]]
```

$\frac{b\,x\,\sqrt{a^2 - x^2}}{2\,a} + \frac{1}{2}a\,b\,\text{ArcSin}\left[\frac{x}{a}\right]$

Но

```
SBOKM/.{b/a √(a^2 - x^2) → y}
```

$\frac{x\,y}{2}+\frac{1}{2}a\,b\,\text{ArcSin}[\frac{x}{a}]$

Так как первое слагаемое представляет площадь ΔOKM, то, отнимая ее, для площади сектора получим выражение $S_{OMB}=\frac{ab}{2}\arcsin\frac{x}{a}$. При $x=a$ для площади четверти эллипса находим значение $\frac{\pi ab}{4}$.

Площадь области D на плоскости xOy, заданной неравенствами или неявно, *можно находить с помощью двойного интеграла*

$$S=\iint\limits_D d\,x\,d\,y\,.$$

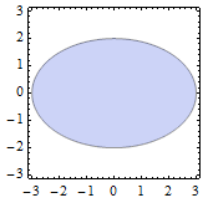
Пример 2. Площадь эллипса $\frac{x^2}{a^2}+\frac{y^2}{b^2}=1$.

$\mathbf{rgn}=\frac{\boldsymbol{x^2}}{\boldsymbol{9}}+\frac{\boldsymbol{y^2}}{\boldsymbol{4}}\leq\mathbf{1}$;

$\mathbf{Integrate}[\mathbf{Boole}[\mathbf{rgn}],\{\boldsymbol{x},-\mathbf{3},\mathbf{3}\},\{\boldsymbol{y},-\mathbf{2},\mathbf{2}\}]$

6π

$\mathbf{RegionPlot}[\mathbf{rgn},\{\boldsymbol{x},-\mathbf{3},\mathbf{3}\},\{\boldsymbol{y},-\mathbf{3},\mathbf{3}\}]$



Пример. Площадь повернутого эллипса $a\,x^2+2b\,x\,y+c\,y^2=1\,(a\,c-b^2>0,c>0)$.

$\mathbf{Clear}[\boldsymbol{a},\boldsymbol{b},\boldsymbol{c}]$;

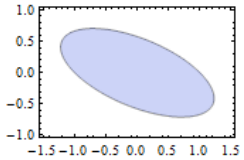
$\mathbf{Integrate}[\mathbf{Boole}[\boldsymbol{ax^2}+\mathbf{2}\boldsymbol{bxy}+\boldsymbol{cy^2}\leq\mathbf{1}],\{\boldsymbol{x},-\infty,\infty\},\{\boldsymbol{y},-\infty,\infty\},$
$\mathbf{Assumptions}:\to\{\boldsymbol{c}>0,\boldsymbol{ac}-\boldsymbol{b^2}>0\}]$

$\frac{\pi}{\sqrt{-b^2+ac}}$

Изобразим эллипс при некоторых значениях параметров a, b, c.

$\boldsymbol{a}=\mathbf{1};\boldsymbol{b}=\mathbf{1};\boldsymbol{c}=\mathbf{3}$;

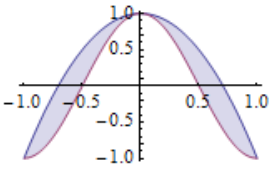
$\mathbf{RegionPlot}[\boldsymbol{ax^2}+\mathbf{2}\boldsymbol{bxy}+\boldsymbol{cy^2}\leq\mathbf{1},\{\boldsymbol{x},-\mathbf{1.5},\mathbf{1.5}\},\{\boldsymbol{y},-\mathbf{1.},\mathbf{1.}\},$
$\mathbf{AspectRatio}\to\mathbf{Automatic}]$



Площадь между двумя кривыми в *Mathematica* можно вычислить несколькими способами: как однократный интеграл от их разности; как повторный интеграл по области, ограниченной этими кривыми; как двойной интеграл по области.

Пример 1. Вычислить площадь между кривыми $y_1(x)=1-2x^2$ и $y_2(x)=\cos\pi x$.

$\mathbf{Plot}[\{\mathbf{1}-\boldsymbol{2x^2},\mathbf{Cos}[\boldsymbol{\pi x}]\},\{\boldsymbol{x},-\mathbf{1},\mathbf{1}\},\mathbf{Filling}\to\mathbf{1}\to\{\mathbf{2}\}]$



Вычислим площадь между кривыми как однократный интеграл от их разности

```
Integrate[1 - 2x^2 - Cos[πx], {x, -1, 1}]
```

$\frac{2}{3}$

Вычислим площадь между кривыми как повторный интеграл по области, ограниченной этими кривыми

```
Integrate[1, {x, -1, 1}, {y, Cos[πx], 1 - 2x^2}]
```

$\frac{2}{3}$

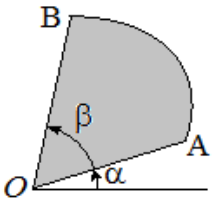
Вычислим площадь между кривыми как двойной интеграл

```
Integrate[Boole[Cos[πx] < y < 1 - 2x^2], {x, -1, 1}, {y, -1, 1}]
```

```
2 / 3
```

Площадь области в полярных координатах.

Дан сектор AOB, ограниченный кривой AB и двумя радиусами – векторами OA и OB (каждый из которых может свестись к точке). Кривая задается полярным уравнением $r = r(\varphi)$ $(\alpha \le \varphi \le \beta)$.



Площадь криволинейного сектора AOB вычисляется по формуле

$$S = \iint\limits_D d\,s = \iint\limits_D r\,d\,r\,d\,\varphi = \int\limits_\alpha^\beta d\,\varphi \int\limits_0^{r(\varphi)} r\,d\,r\,, \tag{5}$$

Если вычислить внутренний интеграл, то получим

$$S = \frac{1}{2}\int\limits_\alpha^\beta r^2(\varphi)d\,\varphi\,. \tag{6}$$

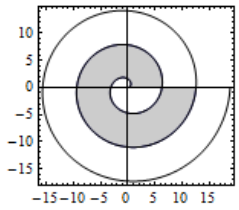
Пример. Вычислить площадь фигуры, заключенной между последовательными витками (первым и вторым) архимедовой спирали $r = a\varphi$. Очевидно, что она вычисляется как разность площадей второго и первого витков.

```
a =.;
r[φ_] = aφ;
s1 = Integrate[r[φ]^2, {φ, 0, 2π}]/2;     (* площадь 1-го витка *)
s2 = Integrate[r[φ]^2, {φ, 2π, 4π}]/2;   (* площадь 2-го витка *)
s2 - s1
```

$8a^2\pi^3$

Поскольку у функции `PolarPlot` нет опции `Filling`, то для представления области перейдем к ее параметрическому представлению

```
a = 1; r[φ_] = a φ;
rv1[t_] = {r[t] Cos[t], r[t] Sin[t]};  (* радиус-вектор первого витка спирали *)
rv2[t_] = rv1[t + 2π];                 (* радиус-вектор второго витка спирали *)
pa = ParametricPlot[{v rv1[t] + (1 − v)rv2[t]}, {t, 0, 2π}, {v, 0, 1},
                    Mesh → None];
pc = PolarPlot[r[φ], {φ, 0, 6π}];
Show[pa, pc]
```

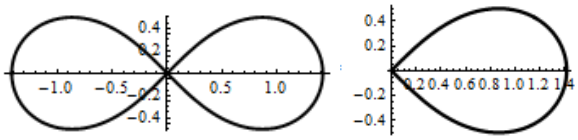


Заметим, что площадь **s1**, посчитанная выше, соответствует незакрашенной внутренней области рисунка.

□

Пример. Найти площадь лемнискаты $r^2 = 2a^2 Cos\,2\varphi$.

```
a = 1; r[φ_] = a√2 √Cos[2φ];
PolarPlot[r[φ], {φ, −π, π}]                    (* рисунок слева *)
PolarPlot[r[φ], {φ, −π/4, π/4},
          PlotStyle → {Black, Thickness[0.02]}]   (* рисунок справа *)
```



Правый рисунок показывает, что достаточно удвоить площадь правого овала, которому отвечает диапазон изменения угла $-\frac{\pi}{4} \le \varphi \le \frac{\pi}{4}$. Имеем

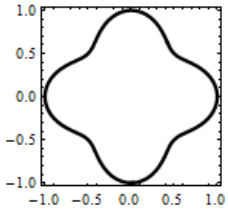
```
a =.; r[φ_] = a√2Cos[2φ];
s1 = Integrate[r[φ]^2, {φ, −π/4, π/4}]/2
```

a^2

Т.о. площадь лемнискаты (двух петель) равна $2a^2$.

Пример. Вычислим площадь фигуры, ограниченной кривой $\left(x^2 + y^2\right)^3 = a^2\left(x^4 + y^4\right)$. Легко видеть, что кривая симметрична относительно осей и точка $(0,0)$ является изолированной.

```
a = 1;
ContourPlot[(x^2 + y^2)^3 == a^2(x^4 + y^4), {x, −1, 1}, {y, −1, 1}]
```



Вычисление площади как интеграла по области не дает точного результата.

Integrate[Boole[$(x^2+y^2)^3 <= a^2(x^4+y^4)$],{$x,-\infty,\infty$},{$y,-\infty,\infty$}]

$$\int_{-1}^{1}(-\text{Root}[-x^4+x^6+3x^4\#1^2+(-1+3x^2)\#1^4+\#1^6\&,1]+\text{Root}[-x^4+x^6+3x^4\#1^2+(-1+3x^2)\#1^4+\#1^6\&,2])\,dx$$

Хотя численное значение легко находится

N[%]

```
2.3561945
```

Переход к полярным координатам позволяет получить точный результат. Полярное уравнение кривой имеет вид $r^2 = a^2\left(\cos^4\varphi+\sin^4\varphi\right)$. Учитывая симметрию и используя формулу (5), получаем

a =.; r =.;

S = 8 Integrate[r,{φ,0,π/4},{r,0,a√(Cos[φ]^4+Sin[φ]^4)}]

$$\frac{3a^2\pi}{4}$$

Площадь плоской области, контур которой задан параметрически.

Площадь области D, ограниченной кусочно – гладкой замкнутой кривой L, можно вычислять с помощью следующих криволинейных интегралов:

$$S=\oint_L x\,d\,y, \tag{7}$$

$$S=-\oint_L y\,d\,x. \tag{8}$$

Чаще для вычисления площади применяют более симметричную формулу

$$S=\frac{1}{2}\oint_L x\,d\,y-y\,d\,x \tag{9}$$

Во всех формулах выбирается положительное направление обхода контура *L*.

Для проверки формул (7) – (9) вспомним формулу Грина. Если функции $P=P(x,y), Q=Q(x,y)$ определены в области *D* и имеют непрерывные частные производные $\frac{\partial P}{\partial y}, \frac{\partial Q}{\partial x}$, то имеет место равенство

$$\iint_D\left(\frac{\partial Q}{\partial x}-\frac{\partial P}{\partial y}\right)d\,x\,d\,y=\oint_L P\,d\,x+Q\,d\,y$$

Если $Q=x, P=0$, то получим $\iint_D dx\,dy=\oint_L x\,dy$. Но слева стоит двойной интеграл по области, который представляет ее площадь S. Аналогично, полагая $Q=0, P=-y$ или $Q=x, P=-y$, приходим к (8) и (9).

Пусть теперь параметрическое уравнение кривой *L*, являющейся границей области *D*, имеет вид $x=x(t), y=y(t), \ t_{\min}\le t\le t_{\max}$, тогда формула (7) принимает вид

$$S=\int_{t_{\min}}^{t_{\max}} x(t)y'(t)d\,t \tag{10}$$

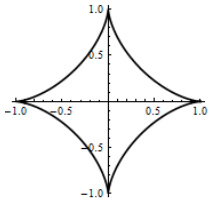
Аналогично записываются другие формулы для площади области, если контур L задан в параметрическом виде.

Пример. Вычислим площадь эллипса.

```
{x, y} = {a Cos[t], b Sin[t]};
Integrate[xD[y, t], {t, 0, 2π}]
a b π
a = 3; b = 2;
ParametricPlot[{x, y}, {t, 0, 2π}]
```

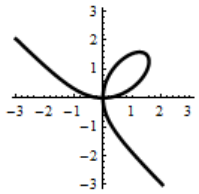
Пример. Вычислим площадь астроиды по формуле (9)

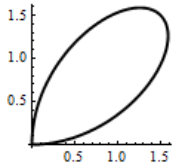
```
x[t_] = a Cos[t]^3;
y[t_] = a Sin[t]^3;
x[t]Dt[y[t], t, Constants → a]
y[t]Dt[x[t], t, Constants → a]
  3a^2Cos[t]^4Sin[t]^2
-3a^2Cos[t]^2Sin[t]^4
S = 1/2 Integrate[x[t]Dt[y[t], t, Constants → a] -
                  y[t]Dt[x[t], t, Constants → a], {t, 0, 2π}]
3a^2π
-----
  8
a = 1;  ParametricPlot[{x[t], y[t]}, {t, 0, 2π}]
```



Пример. Найти площадь петли декартова листа $x^3 + y^3 = 3a\,x\,y$.

```
a = 1;
ContourPlot[x^3 + y^3 == 3 a x y, {x, -3, 3}, {y, -3, 3},
            Axes → True, Frame → False]     (* след. рисунок слева *)
```





Параметрические уравнения контура петли можно записать в виде $x = \dfrac{2at}{1+t^3}$, $y = \dfrac{2at^2}{1+t^3}$. Следующее построение показывает, что петля описывается при изменении параметра t от 0 до ∞.

```
x[t_] = 3at/(1 + t^3);
```

$y[t\_] = \frac{3at^2}{1+t^3}$;
ParametricPlot[{$x[t], y[t]$},{$t, 0, 100$}] (* предыдущий рисунок справа *)
Для вычисления площади используем формулу (9). Имеем
$a =.$; $x[t\_] = \frac{3\,a\,t}{1+t^3}$; $y[t\_] = \frac{3\,a\,t^2}{1+t^3}$;
Simplify[**Integrate**[$x[t]D[y[t], t] - y[t]D[x[t], t]$,{$t, 0, \infty$}]]/2
$\frac{3a^2}{2}$
Отметим, что здесь мы использовали несобственный интеграл с бесконечными пределами.

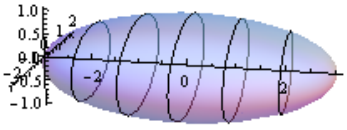
Объем тела.
Объем тела по площади параллельных сечений.

Рассмотрим некоторое тело (V), содержащееся между плоскостями $x=a$ и $x=b$, и станем рассекать его плоскостями, перпендикулярными к оси x. Допустим, что все сечения квадрируемы, и пусть площадь сечения $P(x)$, отвечающего абсциссе x, будет непрерывной функцией от x (для $a \le x \le b$). Пусть также любые два различных сечения, будучи спроектированы на плоскость, перпендикулярную к оси x, оказываются всегда содержащимися одно в другом. Тогда тело (V) имеет объем, который выражается формулой

$$V = \int_a^b P(x)\,dx \qquad (11)$$

Пример. Найти объем трехосного эллипсоида $\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1$.

Плоскость, перпендикулярная к оси x и проходящая через точку $M(x,0,0)$, пересечет эллипсоид по эллипсу (см. следующий рисунок).
$a = 3; b = 2; c = 1;$
$x[u\_, v\_] = a\,\mathbf{Cos}[u]\,\mathbf{Cos}[v];$
$y[u\_, v\_] = b\,\mathbf{Cos}[u]\,\mathbf{Sin}[v];$
$z[u\_, v\_] = c\,\mathbf{Sin}[u];$
ParametricPlot3D[{$x[u, v], y[u, v], z[u, v]$},{$u, -\pi/2, \pi/2$},{$v, 0, 2\pi$},
Boxed → **False**, **AxesOrigin** → {$-a, 0, 0$}, **PlotStyle** → **Opacity**[0.5],
MeshFunctions → {#1&}, **Mesh** → 5]



Уравнение проекции этого эллипса на плоскость yOz будет таково

$$\frac{y^2}{b^2\left(1 - x^2/a^2\right)} + \frac{y^2}{c^2\left(1 - x^2/a^2\right)} = 1 \qquad (x = const).$$

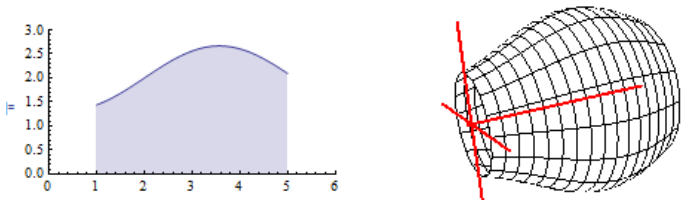
Отсюда ясно, что полуоси его будут, соответственно $b\sqrt{1-x^2/a^2}$ и $c\sqrt{1-x^2/a^2}$, а площадь выразится так $P(x)=\pi bc\left(1-x^2/a^2\right)$. Таким образом, имеем

```
Clear[a, b, c];
P[x_] = (π b c)/a^2 (a^2 - x^2);
V = ∫_-a^a P[x] 𝕕x
```

$$\frac{4}{3}\,a\,b\,c\,\pi$$

Объем тела вращения.

Частный случай формулы (11), когда заведомо выполняется указанное выше предположение о взаимном расположении сечений, представляют тела вращения. Пусть на плоскости xOy задана кривая уравнением $y=f(x)$ $(a\le x\le b)$, где $f(x)$ непрерывна и неотрицательна. Станем вращать ограниченную ею криволинейную трапецию вокруг оси x (см. следующий рисунок).



Сечения полученного тела (V) удовлетворяют указанному предположению, поскольку проектируются на плоскость yOz в виде концентрических кругов. Здесь $P(x)=\pi y^2=\pi\left(f(x)\right)^2$, так что

$$V=\pi\int_a^b y^2dx=\pi\int_a^b\left(f(x)\right)^2dx \qquad (12)$$

Выражение $V=\pi\int_a^b y^2dx$ можно использовать и в случае, когда кривая задана параметрически.

Если криволинейная трапеция ограничена снизу и сверху кривыми $y_1=f_1(x)$ и $y_2=f_2(x)$, то, очевидно

$$V=\pi\int_a^b\left(y_2{}^2-y_1{}^2\right)dx=\pi\int_a^b\left(\left(f_2(x)\right)^2-\left(f_1(x)\right)^2\right)dx, \qquad (13)$$

хотя предположение о взаимном расположении сечений в этом случае может нарушаться.

Пример. Пусть эллипс $\frac{x^2}{a^2}+\frac{y^2}{b^2}=1$ вращается вокруг оси x. Тогда, так как $y^2=\frac{b^2}{a^2}\left(a^2-x^2\right)$, для объема эллипсоида вращения имеем

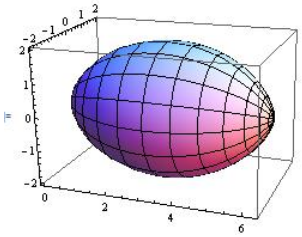
$$V = \pi \int_{-a}^{a} \frac{b^2}{a^2}(a^2 - x^2)\, dx$$

$$\frac{4}{3} a\, b^2\, \pi$$

□

Пример. Определим объем тела, полученного вращением ветви циклоиды $x = a(t - \sin t)$, $y = a(1 - \cos t)$ $(0 \le t \le 2\pi)$ вокруг оси *Ox*. Вначале изобразим тело.

```
a = 1; xc[t_] = a(t - Sin[t]); yc[t_] = a(1 - Cos[t]);
RevolutionPlot3D[{xc[u], yc[u]}, {u, 0, 2π}, RevolutionAxis → {1, 0, 0}]
```



Используем первую форму выражения (12)

```
a =.; x[t_] = a(t - Sin[t]); y[t_] = a(1 - Cos[t]);
V = π Integrate[y[t]^2 D[x[t], t], {t, 0, 2π}]
```

$$5a^3\pi^2$$

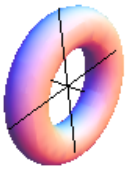
□

Пример. Определим объем тора

$$(x^2 + y^2 + z^2 + R^2 - r^2)^2 - 4R^2(y^2 + z^2) = 0$$

Изобразим тело.

```
R = 3; r = 1;
RegionPlot3D[(x^2 + y^2 + z^2 + R^2 - r^2)^2 - 4R^2(y^2 + z^2) ≤ 0,
    {x, -r, r}, {y, -(R + r), (R + r)}, {z, -(R + r), (R + r)},
     BoxRatios → {2r, 2(R + r), 2(R + r)}, Mesh → None,
     Boxed → False, AxesOrigin → {0, 0, 0}, Ticks → None]
```



Объем тора вычислим как разность объемов, образованных вращением вокруг оси *Ox* верхней и нижней полуокружностей $y = R \pm \sqrt{r^2 - x^2}$, представляющих сечение тора плоскостью *xOy* . Здесь мы воспользуемся формулой (13).

```
Clear[r, R];
V = π Integrate[(R + √(r^2 - x^2))^2 - (R - √(r^2 - x^2))^2, {x, -r, r},
                                                  Assumptions :→ R > r > 0]
```

$$2\pi^2 r^2 R$$

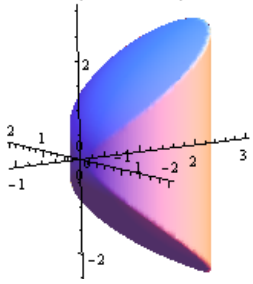
Вычисление объема с помощью тройного интеграла.

Объем трехмерного тела (V) можно вычислять как интеграл по области

$$V = \iiint\limits_{(V)} d\,x\,d\,y\,d\,z \qquad (14)$$

Пример. Найти объем V тела, вырезанного цилиндром $x^2 + y^2 = 2\,a\,x$ из параболоида вращения $y^2 + z^2 = 4a\,x$. Вначале изобразим тело (V).

```
a = 1;
RegionPlot3D[x^2 + y^2 ≤ 2ax&&y^2 + z^2 ≤ 4ax,
    {x, −1, 3}, {y, −2, 2}, {z, −3, 3}, PlotPoints → 100,
    Boxed → False, AxesOrigin → {0, 0, 0}, Mesh → None]
```



Теперь находим объем V.

```
a =.;
Integrate[Boole[x^2 + y^2 ≤ 2 a x && y^2 + z^2 ≤ 4 a x],
    {x, −∞, ∞}, {y, −∞, ∞}, {z, −∞, ∞}, Assumptions :→ a > 0]
```

$$\frac{2}{3}a^3(8 + 3\pi)$$

Вычисление объема с помощью двойного интеграла.

Объем вертикального цилиндрического тела, имеющего своим основанием область D на плоскости xOy и ограниченного сверху поверхностью $z = f(x, y)$, выражается двойным интегралом

$$V = \iint\limits_D z\,d\,x\,d\,y = \iint\limits_D f(x, y)\,d\,x\,d\,y \qquad (15)$$

Пример. Найти объем V тела, вырезанного цилиндром $x^2 + y^2 = R\,x$ из сферы $x^2 + y^2 + z^2 = R^2$. Выполним вычисление по формуле (15). Имеем

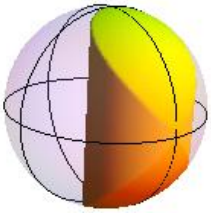
```
R =.;  z[x_, y_] = √(R^2 − x^2 − y^2);
V = 2 Integrate[z[x, y] Boole[x^2 + y^2 ≤ Rx], {x, −∞, ∞}, {y, −∞, ∞},
                Assumptions :→ R > 0]
```

$$\frac{2}{9}(-4 + 3\pi)R^3$$

Теперь изобразим тело (V).

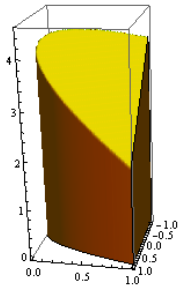
```
R = 1;
p1 = RegionPlot3D[x^2 + y^2 ≤ R x && x^2 + y^2 + z^2 ≤ R^2,
    {x, −R, R}, {y, −R, R}, {z, −R, R}, PlotPoints → 100,
    Mesh → None, PlotStyle → Yellow];
p2 = RegionPlot3D[x^2 + y^2 + z^2 ≤ R^2,
```

```
    {x, -R, R}, {y, -R, R}, {z, -R, R}, PlotPoints → 50,
     Mesh → {1, 1, 1}, PlotStyle → Opacity[0.2]];
Show[p2, p1, Boxed → False, Axes → None, BoxRatios-> {1, 1, 1}]
```



Пример. Найти объем V вертикального цилиндра, который сверху ограничен частью плоскости $x+y+z=4$, снизу – частью плоскости *xOy*, заключенной между параболой $y=x^2$ и прямой $y=1$. Сперва изобразим тело.

```
RegionPlot3D[x + y + z ≤ 4 && y ≥ x^2 && y ≤ 1,
    {x, -1, 1}, {y, 0, 1}, {z, 0, 4.5}, PlotPoints → 100,
     Mesh → None, PlotStyle → Yellow, BoxRatios → {2, 2, 4.5}]
```



Теперь вычисляем объем.

```
f[x_, y_] = 4 - x - y;
V = Integrate[f[x, y]Boole[x^2 ≤ y ≤ 1], {x, -1, 1}, {y, 0, 1}]
```

$$\frac{68}{15}$$

Площадь поверхности тела.

Площадь поверхности, заданной явным уравнением.

Пусть поверхность задана явным уравнение $z=f(x,y)$. Площадь куска этой поверхности, который проектируется на плоскость *xOy* в виде замкнутой области *D*, вычисляется по формуле

$$S=\iint_D \sqrt{1+f_x^2+f_y^2}\,dx\,dy \qquad (16)$$

Пример. Вычислим площадь поверхности сферы. Уравнение $z=\sqrt{R^2-x^2-y^2}$ представляет полусферу радиуса R. Учитывая, что (16) даст площадь только верхней полусферы, имеем

```
z = Sqrt[R^2 - x^2 - y^2];
ds = Simplify[Sqrt[1 + D[z, x]^2 + D[z, y]^2], Assumptions :→ R > 0]
S = 2 Integrate[dsBoole[x^2 + y^2 ≤ R^2], {x, -∞, ∞}, {y, -∞, ∞},
                                          Assumptions :→ R > 0]
```

$4\pi R^2$

Это совпадает с известной формулой элементарной геометрии.

□

Пример. Вычислим площадь среза кругового цилиндра $x^2+y^2\le R^2$ плоскостью $a\,x+b\,y+c\,z+d=0$

```
a = 1; b = 1; c = 1; d = -3; R = 1;
RegionPlot3D[x^2 + y^2 <= R^2 && z >= 0 && a x + b y + c z + d <= 0,
             {x, -R, R}, {y, -R, R}, {z, 0, 5}, PlotPoints → 100,
              BoxRatios → {2R, 2R, 5}, Mesh → None, Boxed → False,
              AxesOrigin → {0, 0, 0}, Ticks → None]
```

```
Clear[x, y, z, R, a, b, c, d];
z = (d - a x - b y)/c;
ds = Sqrt[1 + D[z, x]^2 + D[z, y]^2]
S = Integrate[ds Boole[x^2 + y^2 <= R^2], {x, -∞, ∞}, {y, -∞, ∞},
              Assumptions :→ R > 0&&c > 0]
```

$$\frac{\sqrt{a^2+b^2+c^2}\,\pi\,R^2}{c}$$

□

Если поверхность задана в векторном виде $\mathbf{r}=\mathbf{r}(u,v)$, то площадь участка поверхности вычисляется по формуле

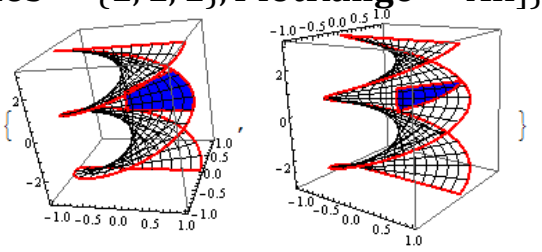
$$S=\iint\limits_D \sqrt{E\,G-F^2}\,d\,u\,d\,v, \qquad (17)$$

где D – область в плоскости изменения параметров u,v, а величины $E=\mathbf{r}_u^2$, $F=\mathbf{r}_u\,\mathbf{r}_v$, $G=\mathbf{r}_v^2$ являются коэффициентами первой квадратичной формы поверхности.

Пример. На прямом геликоиде $x=u\cos v$, $y=u\sin v$, $z=a\,v$ найти площадь четырехугольника, ограниченного линиями $u=0, u=a, v=0, v=1$, а также площадь криволинейного треугольника $0\le u\le a\,v, 0\le v\le 1$. Вначале изобразим эти области.

```
Clear[r, u, v];
a = 1; r = {u Cos[v], u Sin[v], a v};
p1 = ParametricPlot3D[r, {u, -1, 1}, {v, -π, π}, PlotStyle → None,
                      Mesh → {11, 21}, BoundaryStyle → {Red, Thick}];
p2 = ParametricPlot3D[r, {u, 0, a}, {v, 0, 1}, Mesh → None,
                      PlotStyle → Blue, BoundaryStyle → {Red, Thick}];
p3 = ParametricPlot3D[r, {v, 0, 1}, {u, 0, a v}, Mesh → None,
                      PlotStyle → Blue, BoundaryStyle → {Red, Thick}];
```

```
{Show[p1, p2, BoxRatios → {1, 1, 1}, PlotRange → All],
 Show[p3, p1, BoxRatios → {1, 1, 1}, PlotRange → All]}
```



Теперь вычисляем площади областей.

```
Clear[r, ru, rv, a];
r = {u Cos[v], u Sin[v], a v};
ru = D[r, u]; rv = D[r, v];
g11 = Simplify[ru.ru];
g12 = Simplify[ru.rv];
g22 = Simplify[rv.rv];
ds = Simplify[Sqrt[g11g22 − g12^2], Assumptions :→ a > 0];
S1 = Integrate[ds, {u, 0, a}, {u, 0, 1}, Assumptions :→ a > 0]
S2 = Integrate[ds, {v, 0, 1}, {u, 0, av}, Assumptions :→ a > 0];
Simplify[S2/. ArcSinh[x_] → Log[x + Sqrt[x^2 + 1]]]
```

$$\frac{1}{2}a(\sqrt{1+a^2}+a^2\text{Log}[\frac{1+\sqrt{1+a^2}}{a}])$$

$$-\frac{1}{6}a^2(-2+\sqrt{2}-3\text{Log}[1+\sqrt{2}])$$

При вычислении площади S_2 криволинейного треугольника мы предпочли сделать замену $\operatorname{arcsinh} x = \ln\left(x+\sqrt{x^2+1}\right)$.

Дадим пояснения к подстановке ArcSinh[x_] → Log[$x + \sqrt{x^2+1}$]. Когда в выражении `expr` выполняется подстановка `expr/.f[x]→g[x]`, то `f[x]` означает, что функция `f` вычисляется при некотором конкретном значении аргумента x и, если в `expr` находится точное соответствие `f[x]`, то оно заменяется на `g[x]`. Выражения `f` с другими аргументами не заменяются! Подстановка `f[x_]→g[x]` соответствует случаю произвольного аргумента у функции `f`. Поэтому применение правила замены `f[x]→g[x]` и `f[x_]→g[x]` к выражению `f[x]+f[y]+f[z]` приводит к разным результатам.

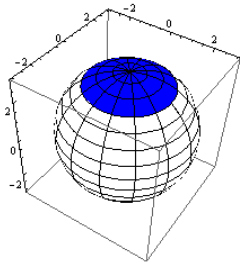
```
f[x] + f[y] + f[z]/. f[x] → g[x]
f[y] + f[z] + g[x]
f[x] + f[y] + f[z]/. f[x_] → g[x]
g[x] + g[y] + g[z]
```

В коде, приведенном выше, для замены мы использовали подстановку с шаблоном `f[x_]→g[x]`, где в качестве функции `f` выступал арксинус гиперболический, а в качестве `g[x]` – выражение Log[$x + \sqrt{x^2+1}$] .

□

Пример. На сфере $\boldsymbol{r} = \left\{R\,\mathrm{Sin}\left[\frac{u}{R}\right]\mathrm{Cos}[v],\ R\mathrm{Sin}\left[\frac{u}{R}\right]\mathrm{Sin}[v],\ R\mathrm{Cos}\left[\frac{u}{R}\right]\right\}$ ($0 \le u \le \pi R$ и $0 \le v < 2\pi$) вычислить площадь круга радиуса r_0 (т.е. $0 \le u \le r_0$). Вначале нарисуем область «круга» на сфере.

```
Remove[r]; R = 3; r0 = 2;
r = {R Sin[u/R] Cos[v], R Sin[u/R] Sin[v], R Cos[u/R]};
sph = ParametricPlot3D[r, {u, 0, πR}, {v, 0, 2π}, Mesh → {11, 13, 0},
        Lighting → {{Ambient, White}}, BoundaryStyle → Black,
        RegionFunction → ((#4 > r0)&)];
crc = ParametricPlot3D[r, {u, 0, r0}, {v, 0, 2π}, Mesh → {2, 13, 0},
        PlotStyle → Blue];
Show[sph, crc, PlotRange → All]
```



Обратите внимание, что для наложения «круга» на сферу мы из нее исключили область этого «круга».

Теперь вычисляем площадь «круга» по формуле (17).

```
Clear[R, r, ru, rv, r0];
r = {R Sin[u/R] Cos[v], R Sin[u/R] Sin[v], R Cos[u/R]};
ru = D[r, u]; rv = D[r, v];
g11 = Simplify[ru.ru];
g12 = Simplify[ru.rv];
g22 = Simplify[rv.rv];
ds = Simplify[√(g11g22 − g12^2), Assumptions :→ R > 0 && u > 0];
S = Integrate[ds, {u, 0, r0}, {v, 0, 2π}, Assumptions :→ R > 0 && 0 < r0 < R];
TrigFactor[S]
```

$4\pi R^2 \mathrm{Sin}[\frac{\mathrm{r0}}{2R}]^2$

Если $r_0 = \pi R$, то кругом радиуса r_0 на сфере является сфера. Ее площадь будет $S = 4\pi R^2 \sin\dfrac{\pi R}{2R} = 4\pi R^2$, что совпадает с классической формулой.

□

Площадь поверхности, получаемой вращением кривой $y = f(x)$ $(a \le x \le b)$ *вокруг оси* Ox, вычисляется по формуле

$$S = 2\pi \int_a^b y\sqrt{1 + y'^2_x}\,dx = 2\pi \int_a^b f(x)\sqrt{1 + (f'(x))^2}\,dx \qquad (18)$$

Пример. Вычислить площадь поверхности шарового пояса. Пусть полуокружность, описанная около начала радиусом r, вращается вокруг оси *x*. Тогда $y = \sqrt{r^2 - x^2}$. Вычисляем

```
r =.; f[x_] = Sqrt[r^2 - x^2];
sr = Simplify[f[x] Sqrt[1 + D[f[x], x]^2], Assumptions :> r > 0 && 0 <= x < r]
S = 2 Pi Integral_x1^x2 sr dx
S /. (x2 - x1) -> h
```

$2\,\pi\,r(-\mathrm{x1} + \mathrm{x2})$

$2\,h\,\pi\,r$

Таким образом, площадь поверхности пояса, описанного дугой, концы которой имеют абсциссы x_1 и x_2, будет равна $2\pi r h$, где h есть высота пояса. В частности при $x_1 = -r$, $x_2 = r$ получаем площадь всей сферы $4\pi r^2$.

□

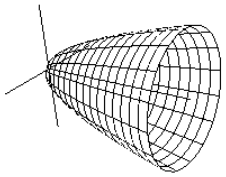
Пример. Вычислить площадь поверхности, образованной вращением вокруг оси О*x* части кривой $y^2 = 4 + x$, отсеченной прямой *x*=2.

Уравнение $y^2 = 4 + x$ определяет параболу с вершиной в точке (-4, 0) и осью симметрии О*x*, поэтому для вычисления площади поверхности вращения достаточно рассмотреть одну ветвь параболы $y = \sqrt{4 + x}$ на отрезке [-4,2].

```
f[x_] = Sqrt[4 + x];
sr = Simplify[f[x] Sqrt[1 + D[f[x], x]^2], Assumptions :> x > -4]
S = 2 Pi Integral_-4^2 sr dx
```

$\frac{62\pi}{3}$

```
RevolutionPlot3D[f[x], {x, -4, 2}, RevolutionAxis -> {1, 0, 0},
    Boxed -> False, AxesOrigin -> {-4, 0, 0},
    Lighting -> {{"Ambient", White}}, Ticks -> None]
```



Пример. Определим площадь поверхности эллипсоида вращения. Если эллипс $\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$ вращается вокруг оси *x*, то для вычисления площади поверхности вращения достаточно рассмотреть одну ветвь эллипса $y = b\sqrt{1 - x^2/a^2}$. Будем также полагать $a > b$. Тогда имеем

```
Clear[a, b];
```

$$f[x\_] = b\sqrt{1-\frac{x^2}{a^2}};$$

$$sr = Simplify[f[x]\sqrt{1+D[f[x],x]^2},\ Assumptions :\to a>0\ \&\&\ b>0\ \&\&\ -a<x<a];$$

$$S = 2\pi\, Integrate[sr,\{x,-a,a\}, Assumptions :\to 0<b<a\ \&\&\ a>b];$$

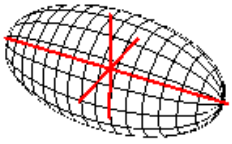
$$S/.(a^2-b^2)\text{^}q\_ \to (a\,p)\text{^}(2q)$$

$$2\,b\,\pi\left(b+\frac{a\,\mathrm{ArcSin}[p]}{p}\right)$$

Здесь $p=\dfrac{\sqrt{a^2-b^2}}{a}$ является эксцентриситетом эллипса.

$$a=2; b=1;$$

$$RevolutionPlot3D[f[x],\{x,-a,a\}, RevolutionAxis\to\{1,0,0\},\ Boxed\to False, AxesOrigin\to\{0,0,0\},\ Lighting\to\{\{Ambient, White\}\}, Ticks\to False,\ AxesStyle\to Directive[Red, Thick]]$$



Если *кривая задана параметрическими уравнениями* $x=x(t),\ y=y(t)\ \ (t_0\le t\le T)$, то *площадь поверхности вращения вокруг оси* X вычисляется по формуле

$$S=2\pi\int_{t_0}^{T} y(t)\sqrt{(x'(t))^2+(y'(t))^2}\,dt \qquad (19)$$

Пример. Вычислим площадь поверхности тора с расстоянием от центра образующей окружности до оси вращения R и с радиусом образующей окружности r $(r<R)$. Эту площадь вычислим как площадь поверхности, образуемой вращением окружности $x=r\cos t,\ \ y=R+r\sin t$ вокруг оси Ox.

$$x[t\_] = rCos[t];$$

$$y[t\_] = R + rSin[t];$$

$$ds = Simplify[y[t]\sqrt{D[x[t],t]^2+D[y[t],t]^2},\ Assumptions :\to R>0\ \&\&\ 0<r<R]$$

$$S = 2\pi\, Integrate[ds,\{t,0,2\pi\}, Assumptions :\to 0<r<R\ \&\&\ R>0]$$

$$4\pi^2 rR$$

Это можно представить как $S=2\pi r\cdot 2\pi R$. Стало быть, площадь тора равна произведению длины образующей окружности на длину пути, описываемого ее центром.

Пример. Дана циклоида $x=a(t-\sin t),\ \ y=a(1-\cos t)$. Найти площадь поверхности, образованной вращением ее вокруг оси Ox.

$$a=.; x[t\_] = a(t - Sin[t]); y[t\_] = a(1 - Cos[t]);$$

```
ds = Simplify[y[t]√(D[x[t], t]^2 + D[y[t], t]^2], Assumptions :> a > 0]
S = 2 π Integrate[ds, {t, 0, 2π}, Assumptions :> a > 0]
```

$$\frac{64a^2\pi}{3}$$

Для *полярной системы координат*, если вращение кривой $\rho = \rho(\varphi)$ производится вокруг полярной оси, *площадь поверхности тела вращения* вычисляется по формуле

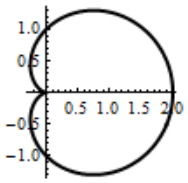
$$S = 2\pi \int_{\alpha}^{\beta} \rho(\varphi)\sin\varphi \sqrt{\rho^2(\varphi) + (\rho'(\varphi))^2}\, d\varphi \qquad (20)$$

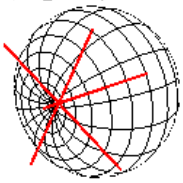
Пример. Найти площадь поверхности, образованной вращением кардиоиды $r = a(1+\cos\varphi)$ вокруг полярной оси. Поскольку кардиоида симметрична относительно полярной оси, то пределами интегрирования в формуле (20) нужно выбрать 0 и π. Имеем

```
a =.; r[φ_] = a(1 + Cos[φ]);
ds = Simplify[r[t]Sin[t]√(r[t]^2 + D[r[t], t]^2], Assumptions :> a > 0];
S = 2 π Integrate[ds, {t, 0, π}, Assumptions :> a > 0]
```

$$\frac{32a^2\pi}{5}$$

```
a = 1;
PolarPlot[r[φ], {φ, 0, 2π}, PlotStyle → {Black, Thick}] (* рис. слева *)
RevolutionPlot3D[{r[t]Cos[t], r[t]Sin[t]}, {t, 0, π},
    RevolutionAxis → {1, 0, 0}, Boxed → False, AxesOrigin → {0, 0, 0},
    Lighting → {{Ambient, White}}, Ticks → False,
    AxesStyle → Directive[Red, Thick]]   (* рисунок справа *)
```





3.4 Векторный анализ

Векторный анализ изучает скалярные и векторные поля. Говорят, что в некоторой области задано поле, если каждой точке этой области соответствует определенное значение некоторой величины – числовой или векторной.

Если в каждой точке области задана величина, принимающая числовые значения, то поле называется скалярным, если же в каждой точке области задан вектор, то поле называется векторным.

Если поле является скалярным, то задание поля равносильно заданию функции трех переменных $u(x, y, z)$ (или двух, если поле плоское). Если поле векторное, то надо задать все проекции переменного вектора на оси координат $\vec{\mathbf{A}} = P(x, y, z)\mathbf{i} + Q(x, y, z)\mathbf{j} + R(x, y, z)\mathbf{k}$, где P, Q, R – скалярные функции.

Векторный анализ является основой многих физических и математических моделей и, естественно, что в системе *Mathematica* имеются функции, выполняющие основные операции векторного анализа. При этом понятия «вектор» в *Mathematica* нет. Вместо этого используется понятие списка, который может быть одномерным (аналог вектора), двумерным (аналог матрицы) и многомерным. Для многих функций, рассматриваемых в данном разделе, аргументами могут быть многомерные списки. Однако мы уделим внимание в основном одномерным спискам, представляющим вектора и вектор – функции.

3.4.1 Алгебраические операции с векторами

Поскольку векторный анализ имеет дело с семействами функций (в терминологии *Mathematica* со списками функций), то уместно напомнить о некоторых функциях, выполняющих алгебраические операции с векторами/списками.

Функция `Norm[expr]` возвращает норму числа, вектора или матрицы.

```
Norm[{x, y, z}]
```

$$\sqrt{\text{Abs}[x]^2 + \text{Abs}[y]^2 + \text{Abs}[z]^2}$$

или

```
Simplify[Norm[{x, y, z}],
                Assumptions :→ x ∈ Reals&&y ∈ Reals&&z ∈ Reals]
```

$$\sqrt{x^2 + y^2 + z^2}$$

```
Norm[-3 + 4I]
```

5

```
Norm[{{1, 2}, {-3, 4}}]
```

$$\sqrt{5(3 + \sqrt{5})}$$

В формате `Norm[{x1, x2, ...}, n]` вычисляется выражение $\left(x_1^n + x_2^n + \ldots\right)^{1/n}$. Если $n = \infty$, то вычисляется норма $\max\left(|x_1|, |x_2|, \ldots\right)$.

```
Norm[{x, y, z}, n]
```

$$(\text{Abs}[x]^n + \text{Abs}[y]^n + \text{Abs}[z]^n)^{\frac{1}{n}}$$

```
Norm[{1, 2, 3}, 4]
```

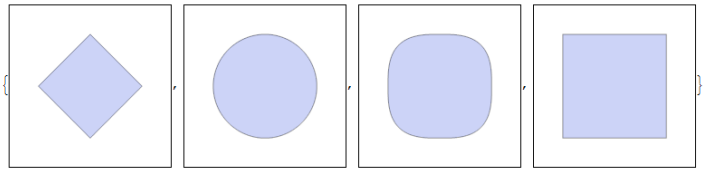
$$2^{1/4}\sqrt{7}$$

```
Norm[{x, y, z}, Infinity]
```

Max[Abs[x], Abs[y], Abs[z]]

Вот графический пример, иллюстрирующий применение функции `Norm`.

```
Table[RegionPlot[Norm[{x, y}, p] ≤ 1, {x, -1.5, 1.5}, {y, -1.5, 1.5},
                FrameTicks → None], {p, {1, 2, 3, Infinity}}]
```



Функция `Normalize[список]` возвращает нормированный вектор, координаты которого изначально были заданы `списком`.

Normalize[{**2**, **3**, **6**}]

$$\left\{\frac{2}{7}, \frac{3}{7}, \frac{6}{7}\right\}$$

Simplify[**Normalize**[{$\boldsymbol{u}+\boldsymbol{v}, \boldsymbol{u}-\boldsymbol{v}$}],
Assumptions :→ $\boldsymbol{u} \in$ **Reals** && $\boldsymbol{v} \in$ **Reals**]

$$\left\{\frac{u+v}{\sqrt{2}\sqrt{u^2+v^2}}, \frac{u-v}{\sqrt{2}\sqrt{u^2+v^2}}\right\}$$

$\boldsymbol{v} = \{\mathbf{1}, \mathbf{2}+\mathbf{3}\boldsymbol{I}, \mathbf{4}\boldsymbol{I}, \mathbf{5}+\mathbf{6}\boldsymbol{I}\}$;

Normalize[$\boldsymbol{v}$]

$$\left\{\frac{1}{\sqrt{91}}, \frac{2+3i}{\sqrt{91}}, \frac{4i}{\sqrt{91}}, \frac{5+6i}{\sqrt{91}}\right\}$$

Функция `Orthogonalize[{v1,v2,...}]` строит нормированный ортогональный базис из векторов `v1,v2,...`.

Orthogonalize[{{**2**, **0**, **0**}, {**2**, **1**, **0**}, {**3**, **2**, **1**}}]

```
{{1,0,0},{0,1,0},{0,0,1}}
```

Те же вектора, но в другой последовательности.

Orthogonalize[{{**2**, **1**, **0**}, {**3**, **2**, **1**}, {**2**, **0**, **0**}}]

$$\left\{\left\{\frac{2}{\sqrt{5}}, \frac{1}{\sqrt{5}}, 0\right\}, \left\{-\frac{1}{\sqrt{30}}, \sqrt{2/15}, \sqrt{5/6}\right\}, \left\{\frac{1}{\sqrt{6}}, -\sqrt{2/3}, \frac{1}{\sqrt{6}}\right\}\right\}$$

Если вектор в списке один, то он просто нормируется.

Orthogonalize[{{**3**, **1**, **1**}}]

$$\left\{\left\{\frac{3}{\sqrt{11}}, \frac{1}{\sqrt{11}}, \frac{1}{\sqrt{11}}\right\}\right\}$$

Если векторов меньше, чем размерность пространства, то выполняется обычная ортогонализация

Orthogonalize[{{**1**, −**1**, **1**}, {**0**, −**1**, −**1**}}]

$$\left\{\left\{\frac{1}{\sqrt{3}}, -\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}\right\}, \left\{0, -\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}\right\}\right\}$$

Если векторов в списке больше, чем размерность пространства, то последние вектора возвращаются нулевыми.

Orthogonalize[{{**1**, −**1**, **1**}, {**0**, −**1**, −**1**}, {**0**, **3**, **1**}, {−**1**, **0**, **0**}}]

$$\left\{\left\{\frac{1}{\sqrt{3}}, -\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}\right\}, \left\{0, -\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}\right\}, \left\{\sqrt{\frac{2}{3}}, \frac{1}{\sqrt{6}}, -\frac{1}{\sqrt{6}}\right\}, \{0,0,0\}\right\}$$

Функцию `Orthogonalize` можно применять к матрицам (спискам списков)

Simplify[**Orthogonalize**[{{$\boldsymbol{x}, -\boldsymbol{y}$}, {$\boldsymbol{x}+\boldsymbol{y}, \boldsymbol{x}-\boldsymbol{y}$}}],
Assumptions :→ ($\boldsymbol{x} \in$ **Reals**&&$\boldsymbol{y} \in$ **Reals**)]

$$\left\{\left\{\frac{x}{\sqrt{x^2+y^2}}, -\frac{y}{\sqrt{x^2+y^2}}\right\}, \left\{\frac{y}{\sqrt{x^2+y^2}}, \frac{x}{\sqrt{x^2+y^2}}\right\}\right\}$$

Функция `Orthogonalize` может использовать свою собственную функцию скалярного произведения, может обрабатывать многомерные списки, имеет две опции. Подробнее с ней вы можете познакомиться по справочной системе.

Функция `Dot` вычисляет скалярное произведение векторов, а также может выполнить матричное умножение списков (матриц) и умножение матрицы на вектор

Dot[{x, y, z}, {a, b, c}]

$ax + by + cz$

Инфиксной формой функции `Dot` является оператор '**.**' (точка). С его помощью мы можем вычислить произведение векторов и/или матриц.

```
{{a,b},{c,d}}.{x,y}
```

```
{ax+by,cx+dy}
```

или

```
{x,y}.{{a,b},{c,d}}
```

```
{ax+cy,bx+dy}
```

В первом случае `{x,y}` выступает, как вектор-столбец, во втором - как вектор-строка. В следующем выражении присутствуют оба типа.

{x, y}.{x, y}

$x^2 + y^2$

Векторное произведение векторов выполняется функцией `Cross`.

```
Cross[{1,2,3},{2,4,5}]
```

```
{-2,1,0}
```

Инфиксной формой функции `Cross` является оператор '**×**' (крест). Он вводится как `Esc - cross - Esc`.

{x, y, z} × {a, b, c}

$\{cy - bz, -cx + az, bx - ay\}$

Функция `EuclideanDistance[u,v]` вычисляет евклидово расстояние между векторами

EuclideanDistance[{a, b, c}, {x, y, z}]

$\sqrt{\text{Abs}[a-x]^2 + \text{Abs}[b-y]^2 + \text{Abs}[c-z]^2}$

Если вы уверены, что координаты векторов вещественные, то можно избавиться от модулей, выполнив замену

EuclideanDistance[{a, b, c}, {x, y, z}]/. (**Abs**[x_]^**2** → x^**2**)

$\sqrt{(a-x)^2 + (b-y)^2 + (c-z)^2}$

Здесь подстановка Abs[x_]^2 → x^2 любой «квадрат модуля выражения» заменяет «квадратом выражения», т.е. выражение вида $Abs[expr]^2$ заменяется выражением $expr^2$ при любом $expr$.

Евклидово расстояние эквивалентно норме разности векторов.

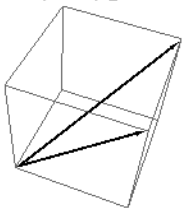
EuclideanDistance[{a, b, c}, {x, y, z}] == **Norm**[{a, b, c} − {x, y, z}]

```
True
```

Функция `VectorAngle[u,v]` вычисляет угол между векторами `u` и `v`.

u = {**1**, **0**, **1**};

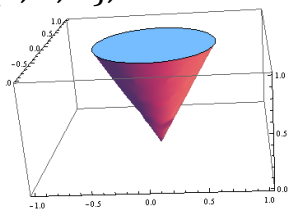
$\mathbf{v} = \{\mathbf{1}, \mathbf{1}, \mathbf{1}\};$
$\mathbf{VectorAngle}[\mathbf{u}, \mathbf{v}]$
$\text{ArcCos}[\sqrt{2/3}]$
$\mathbf{Graphics3D}[\{\mathbf{Thick}, \mathbf{Arrow}[\{\{\mathbf{0}, \mathbf{0}, \mathbf{0}\}, \mathbf{u}\}], \mathbf{Arrow}[\{\{\mathbf{0}, \mathbf{0}, \mathbf{0}\}, \mathbf{v}\}]\}]$



Если координаты векторов вещественные, то выполняя очевидные замены, можно получить классическое выражение для угла между векторами
$\mathbf{VectorAngle}[\{\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}\}, \{\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}\}]/.\{\mathbf{Conjugate}[\mathbf{x\_}] \to \boldsymbol{x}, \mathbf{Abs}[\mathbf{x\_}]\mathbf{\hat{}2} \to \boldsymbol{x}\mathbf{\hat{}2}\}$

$$\text{ArcCos}\left[\frac{a\,x + b\,y + c\,z}{\sqrt{a^2 + b^2 + c^2}\sqrt{x^2 + y^2 + z^2}}\right]$$

Вот один пример использования функции `VectorAngle`.
$\mathbf{RegionPlot3D}[\mathbf{VectorAngle}[\{\mathbf{0}, \mathbf{0}, \mathbf{1}\}, \{\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}\}] < \mathbf{Pi/6}\ \&\&\ \boldsymbol{z} \leq \mathbf{1},$
$\{\boldsymbol{x}, -\mathbf{1}, \mathbf{1}\}, \{\boldsymbol{y}, -\mathbf{1}, \mathbf{1}\}, \{\boldsymbol{z}, \mathbf{0}, \mathbf{1}\}, \mathbf{BoxRatios} \to \{\mathbf{2}, \mathbf{2}, \mathbf{1}\}]$



Функция `Projection[u,v]` вычисляет проекцию вектора `u` на вектор `v`.
$\mathbf{Projection}[\{\mathbf{4}, -\mathbf{6}, \mathbf{3}\}, \{\mathbf{1}, \mathbf{2}, \mathbf{3}\}]$
$\left\{\frac{1}{14}, \frac{1}{7}, \frac{3}{14}\right\}$

Выражение для вычисления проекции имеет вид $\frac{\bar{\mathbf{u}} \cdot \mathbf{v}}{\bar{\mathbf{v}} \cdot \mathbf{v}}\mathbf{v}$, где черта сверху означает комплексное сопряжение. Для вещественных векторов оно совпадает с классическим определением.

Пример. Разложить вектор `u` на составляющие: вдоль вектора `v` и ортогональный к `v` вектор.
$\boldsymbol{u} = \{\mathbf{1}, \mathbf{1}, \mathbf{1}\};$
$\boldsymbol{v} = \{\mathbf{1}, \mathbf{2}, \mathbf{4}\};$
$\mathbf{upar} = \mathbf{Projection}[\boldsymbol{u}, \boldsymbol{v}]$
$\left\{\frac{1}{3}, \frac{2}{3}, \frac{4}{3}\right\}$
$\mathbf{uort} = \boldsymbol{u} - \mathbf{uparallel}$
$\left\{\frac{2}{3}, \frac{1}{3}, -\frac{1}{3}\right\}$

Проверим, что вектора `uort` и `upar` в сумме дают вектор `u` и, что они ортогональны.
$\mathbf{upar} + \mathbf{uort}$
`{1,1,1}`

uort. upar
0

□

В формате `Projection[u,v,f]` вычисляется проекция вектора `u` на вектор `v`, используя в качестве скалярного произведения функцию `f`. Например, если координаты векторов вещественные, то в качестве `f` надо выбирать функцию `Dot`.

Projection[{x, y, z}, {a, b, c}, **Dot**]

$$\left\{\frac{a(ax+by+cz)}{a^2+b^2+c^2}, \frac{b(ax+by+cz)}{a^2+b^2+c^2}, \frac{c(ax+by+cz)}{a^2+b^2+c^2}\right\}$$

Под векторами можно понимать функции. Тогда в качестве скалярного произведения можно использовать определенный интеграл. Например

ip[**f1_, f2_**]:= **Integrate**[**f1** ∗ **f2**, {$x, -\pi/2, \pi/2$}]

Projection[x^2, **Cos**[x], **ip**]

$$\frac{(-8+\pi^2)\text{Cos}[x]}{\pi}$$

Здесь мы создали функцию `ip[f1,f2]`, которая при вычислении проекции используется для вычисления скалярного произведения функций.

Функция `Thread[f[args]]` «проносит» `f` сквозь любой список/вектор, который появляется в аргументе `args`. В результате создается список значений функции `f` на элементах списков из `args`.

Thread[f[{a,b,c}]]

```
{f[a],f[b],f[c]}
```

Thread[f[{a, b, c}, x]]

{$f[a, x], f[b, x], f[c, x]$}

Thread[f[{a, b, c}, {x, y, z}]]

{$f[a, x], f[b, y], f[c, z]$}

Thread[{a,b,c}=={x,y,z}]

```
{a==x,b==y,c==z}
```

Последний результат станет понятен, если аргумент функции `Thread` представить в полной форме

{a, b, c} == {x, y, z}//**FullForm**

Equal[List[a, b, c], List[x, y, z]]

Т.о. функция `Equal` «проносится» внутрь своих списочных аргументов.

В формате `Thread[f[args],h]` функция `Thread` «проносит» `f` сквозь любой объект с заголовком `h`, который появляется в аргументе `args`.

Thread[**Sin**[$x == y$], **Equal**]

Sin[x] == Sin[y]

Рассмотрим теперь преобразование поворота векторов. Оно выполняется функцией `RotationTransform`.

Вначале рассмотрим вектора на плоскости. В формате `RotationTransform[θ]` возвращается матрица преобразования поворота, которую можно применить к любой паре чисел (двумерному вектору или

координатам точки на плоскости). Следует уточнить, что возвращается объект – функция `TransformationFunction`, который действует как функция и который можно применять к одномерным спискам.

RotationTransform[θ]

$$\text{TransformationFunction}\left[\left(\begin{array}{cc|c} \text{Cos}[\theta] & -\text{Sin}[\theta] & 0 \\ \text{Sin}[\theta] & \text{Cos}[\theta] & 0 \\ \hline 0 & 0 & 1 \end{array}\right)\right]$$

Если такую функцию применить к вектору, то мы получим повернутый вектор.

RotationTransform[$\pi/4$][{**1**, **1**}]

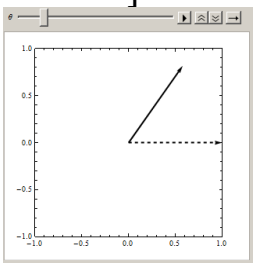
$\{0, \sqrt{2}\}$

RotationTransform[θ][{x, y}]

$\{x\text{Cos}[\theta] - y\text{Sin}[\theta], y\text{Cos}[\theta] + x\text{Sin}[\theta]\}$

В следующем коде мы создаем анимацию вращения вектора. Меняя угол поворота от 0 до 2 π, мы пересчитываем координаты его конца с помощью функции `RotationTransform` для каждого нового значения θ .

```
pnt = {1, 0};
gpstart = Graphics[{Thick, Dashed, Arrow[{{0, 0}, pnt}]}];
Animate[
   pr = RotationTransform[θ][pnt];
   gpr = Graphics[{Thick, Arrow[{{0, 0}, pr}]}];
   Show[gpstart, gpr, Frame → True, PlotRange → {{-1, 1}, {-1, 1}}],
{θ, 0, 2π}, AnimationRunning → False]
```



В формате `RotationTransform[θ,pnt]` возвращается матрица поворота вокруг точки `pnt`. Следующая команда вычисляет координаты точки {*x*,*y*} после поворота ее вокруг точки {1,0} на угол θ.

RotationTransform[θ, {**1**, **0**}][{x, y}]

$\{1 - \text{Cos}[\theta] + x\text{Cos}[\theta] - y\text{Sin}[\theta], y\text{Cos}[\theta] - \text{Sin}[\theta] + x\text{Sin}[\theta]\}$

Сравните с результатом поворота вокруг начала координат.

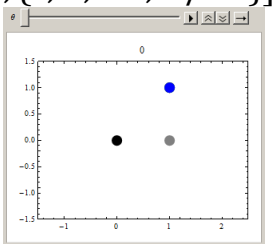
RotationTransform[θ, {**0**, **0**}][{x, y}]

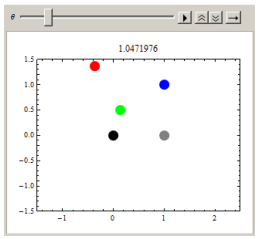
$\{x\text{Cos}[\theta] - y\text{Sin}[\theta], y\text{Cos}[\theta] + x\text{Sin}[\theta]\}$

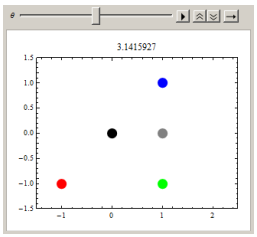
Построим анимацию вращения точки {1,1} одновременно вокруг двух точек: начала координат {0,0} и точки {1,0}.

```
psize = 0.05; v = {1, 1}; pnt = {1, 0};
gpnt0 = Graphics[{Black, PointSize[psize], Point[{0, 0}]}];
gpnt1 = Graphics[{Gray, PointSize[psize], Point[{1, 0}]}];
gpntv = Graphics[{Blue, PointSize[psize], Point[v]}];
Animate[
   vr = RotationTransform[θ][v];
```

gpntvr = Graphics[{Red, PointSize[psize], Point[vr]}];
vp = RotationTransform[θ, pnt][v];
gpntvp = Graphics[{Green, PointSize[psize], Point[vp]}];
Show[gpnt0, gpnt1, gpntvr, gpntvp, gpntv, Frame → True,
PlotRange → {{−1.5, 2.5}, {−1.5, 1.5}}, PlotLabel → θ],
{θ, Table[t, {t, 0, 2π, π/24}]}, AnimationRunning → False]







Здесь на рисунке слева показана панель анимации в нулевой момент времени. Черная точка является началом координат, вокруг которой выполняется одно вращение. Серая точка {1,0} является центром второго вращения. Синяя точка {1,1} является стартовой для обоих вращений. На среднем рисунке показано положение точек после поворота на угол $\pi/3$. Красная точка вращается вокруг начала координат, а зеленая – вокруг точки {1,0}. На правом рисунке показано положение точек после поворота на угол π.

В формате `RotationTransform[θ,u0][u]` выполняется поворот 3D вектора `u` вокруг 3D вектора u_0 на угол θ.

RotationTransform[θ, {0, 0, 1}][{x, y, z}]
{xCos[θ] − ySin[θ], yCos[θ] + xSin[θ], z}

У функции `RotationTransform` имеется еще несколько форматов вызова, с которыми вы можете познакомиться по справочной системе.

Много других типов операций со списками описаны нами в первой части пособия.

3.4.2 Дифференциальные операции векторного анализа

Градиент. Функция `Grad[f,{x1,x2,...,xn}]` вычисляет градиент $\left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n}\right)$. Первым аргументом должно быть выражение `f[x1,x2,...,xn]`, а вторым – список переменных.

Grad[f[x, y, z], {x, y, z}]
{$f^{(1,0,0)}[x, y, z], f^{(0,1,0)}[x, y, z], f^{(0,0,1)}[x, y, z]$}
Grad[$x^2 + y^2 + z^2$, {x, y, z}]
{2x, 2y, 2z}

Можно вычислять градиент функции любого количества переменных (меньше и больше трех).

Grad[f[x, y], {x, y}]
$\{f^{(1,0)}[x, y], f^{(0,1)}[x, y]\}$
Grad[f[x], {x}]
$\{f'[x]\}$

Градиент можно вычислять в различных системах координат. Для этого имеется следующий формат `Grad[f,{x1,x2,...,xn},"система"]`, где третьим аргументом передается строка – имя системы координат.

$\mathbf{Grad}[\boldsymbol{f}[\boldsymbol{r},\boldsymbol{\varphi},\boldsymbol{z}],\{\boldsymbol{r},\boldsymbol{\varphi},\boldsymbol{z}\},\mathbf{"Cylindrical"}]$

$\{f^{(1,0,0)}[r,\varphi,z],\dfrac{f^{(0,1,0)}[r,\varphi,z]}{r},f^{(0,0,1)}[r,\varphi,z]\}$

Первым аргументом может быть список функций. Например, можно вычислить градиент от градиента.

$\boldsymbol{u}=\mathbf{Grad}[\boldsymbol{x}^3+\boldsymbol{y}^3+3\boldsymbol{x}\boldsymbol{y}^2,\{\boldsymbol{x},\boldsymbol{y}\}]$

$\mathbf{Grad}[\boldsymbol{u},\{\boldsymbol{x},\boldsymbol{y}\}]\mathbf{//MatrixForm}$

$\{3x^2+3y^2,6xy+3y^2\}$

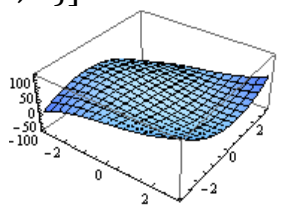
$\begin{pmatrix}6x+4y & 4x+6y\\ 4x+6y & 6x+6y\end{pmatrix}$

Функцию `Grad` можно использовать для определения экстремальных точек функции нескольких переменных. Напомним, что если в стационарной точке функции 2 – х переменных $f'_x=0$, $f'_y=0$ выполняется условие $\Delta=f''_{xx}f''_{yy}-\left(f''_{xy}\right)^2>0$, то она является точкой строгого экстремума, а именно строгого максимума, если в ней $f''_{xx}<0$, и строгого минимума, если $f''_{xx}>0$.

Пример. Исследовать на экстремум функцию $f(x,y)=x^3+3xy^2-15x-12y$.

$\boldsymbol{f}[\mathbf{x\_},\mathbf{y\_}]=\boldsymbol{x}^3+3\boldsymbol{x}\boldsymbol{y}^2-15\boldsymbol{x}-12\boldsymbol{y};$

$\mathbf{Plot3D}[\boldsymbol{f}[\boldsymbol{x},\boldsymbol{y}],\{\boldsymbol{x},-3,3\},\{\boldsymbol{y},-3,3\}]$



Вычисляем градиент функции и находим стационарные точки.

$\mathbf{grad}=\mathbf{Grad}[\boldsymbol{f}[\boldsymbol{x},\boldsymbol{y}],\{\boldsymbol{x},\boldsymbol{y}\}]$

$\mathbf{sol}=\mathbf{Solve}[\mathbf{grad}==\{0,0\},\{\boldsymbol{x},\boldsymbol{y}\}]$

$\{-15+3x^2+3y^2,-12+6xy\}$

$\{\{x\to-2,y\to-1\},\{x\to-1,y\to-2\},\{x\to1,y\to2\},\{x\to2,y\to1\}\}$

Вычисляем матрицу Гессе $H(x)=\begin{pmatrix}f''_{xx} & f''_{xy}\\ f''_{xy} & f''_{yy}\end{pmatrix}$ функции, ее определитель (гессиан) $\Delta=f''_{xx}f''_{yy}-\left(f''_{xy}\right)^2$ и вторые производные $\left(f''_{xx},f''_{yy}\right)$ в стационарных точках.

$\mathbf{gg}=\mathbf{Grad}[\mathbf{grad},\{\boldsymbol{x},\boldsymbol{y}\}];$

$\mathbf{Det}[\mathbf{gg}]/.\mathbf{sol}$

$\boldsymbol{D}[\boldsymbol{f}[\boldsymbol{x},\boldsymbol{y}],\{\boldsymbol{x},2\}]/.\mathbf{sol}$

$\{108,-108,-108,108\}$

$\{-12,-6,\ 6,\ 12\}$

Гессиан Δ положителен в первой и последней точках, следовательно они являются точками экстремума. В первой точке (-2, -1) $f''_{xx}<0$, следовательно,

это точка максимума. В точке (2, 1) $f''_{xx} > 0$ и поэтому она является точкой минимума. Другие две точки не являются точками экстремума.

Заметим, что матрицу Гессе мы вычисляли как градиент от градиента, поскольку градиент можно применять к списку функций.

```
Remove[f]
gg = Grad[Grad[f[x, y], {x, y}], {x, y}];
gg//MatrixForm
```

$$\begin{pmatrix} f^{(2,0)}[x,y] & f^{(1,1)}[x,y] \\ f^{(1,1)}[x,y] & f^{(0,2)}[x,y] \end{pmatrix}$$

□

Напомним, что в декартовой системе координат градиент можно вычислять с использованием функции `D`. В формате `D[f,{{x1,x2,…}}]` вычисляется вектор производных $\left\{\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots\right\}$ скалярной функции `f` (точнее выражения `f[x1,x2,...]`). Но этот вектор и является градиентом функции в декартовой системе.

```
D[f[x, y], {{x, y}}]
```

$\{f^{(1,0)}[x,y], f^{(0,1)}[x,y]\}$

```
D[(x^2 + y^3)^2, {{x, y}}]
```

$\{4x(x^2+y^3), 6y^2(x^2+y^3)\}$

Производная функции `u` по направлению находится как $\langle grad\, u, \vec{\mathbf{e}} \rangle$, где угловые скобки обозначают скалярное произведение векторов, а вектор $\vec{\mathbf{e}}$ является единичным вектором заданного направления.

Пример. Найти производную функции $f(x,y) = x^2 \sin 2y$ в точке $(1, \pi/2)$ в направлении вектора $\mathbf{v} = \left(\frac{3}{5}, -\frac{4}{5}\right)$. Имеем

```
f[x_, y_] = x^2 Sin[2y];
v = {3/5, -4/5};   (* единичный вектор *)
gradf = Grad[f[x, y], {x, y}]/.{x → 1, y → π/2}
```

```
{0, -2}
```

Поскольку v – единичный вектор, то производная по направлению будет

```
dirderiv = v.gradf
```

$$\frac{8}{5}$$

Пример. Найти производную функции $f(x,y,z) = xy + yz + xz$ в точке $(1,1,1)$ в направлении $\mathbf{v} = 2\mathbf{i} + \mathbf{j} - \mathbf{k}$. Имеем

```
F[x_, y_, z_] = xy + yz + xz;
gd = Grad[F[x, y, z], {x, y, z}]/.{x → 1, y → 1, z → 1}
```

```
{2, 2, 2}
```

```
v = {2, 1, -1};
e = Normalize[v]
```

$\left\{\sqrt{2/3}, \frac{1}{\sqrt{6}}, -\frac{1}{\sqrt{6}}\right\}$

dirderiv = $\boldsymbol{v}$.**gd**

4

□

Дивергенция. Функция `Div[{f1,...,fn},{x1,...,xn}]` вычисляет дивергенцию $\frac{\partial f_1}{\partial x_1}+\ldots+\frac{\partial f_n}{\partial x_n}$ вектор – функции `{f1,...,fn}`.

Div[{$\boldsymbol{f}[\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}], \boldsymbol{g}[\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}], \boldsymbol{h}[\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}]$}, {$\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}$}]

$h^{(0,0,1)}[x,y,z] + g^{(0,1,0)}[x,y,z] + f^{(1,0,0)}[x,y,z]$

Div[{$\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}$}, {$\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}$}]

3

Вектор – функции можно присваивать имя и использовать его вместо списка из скалярных функций.

$\boldsymbol{B}$[x_, y_, z_]:= {$\boldsymbol{x^2}, \boldsymbol{y^2}, \boldsymbol{z^2}$}

Div[$\boldsymbol{B}[\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}]$, {$\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}$}]

2x+2y+2z

Дивергенцию можно вычислять в криволинейных системах координат. Для этого имеется следующий формат

`Div[{f1,...,fn},{x1,...,xn},"система"],`

где третьим аргументом передается строка – имя системы координат.

Div[{$\boldsymbol{r}$**Sin**[$\boldsymbol{2\theta}$], $-\boldsymbol{r}$**Cos**[$\boldsymbol{2\theta}$]}, {$\boldsymbol{r},\boldsymbol{\theta}$}, "**Polar**"]

$4\text{Sin}[2\theta]$

Дивергенция вектора любой размерности является скаляром.

Div[{$\boldsymbol{w}$ **Sin**[$\boldsymbol{x}$], $\boldsymbol{w}$ **Cos**[$\boldsymbol{y}$], $\boldsymbol{x}\ \boldsymbol{y}\ \boldsymbol{z}, \boldsymbol{x^2}+\boldsymbol{w^2}$}, {$\boldsymbol{w},\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}$}]

$x\ z + \text{Sin}[x]$

В криволинейной системе координат дивергенция вектора с постоянными компонентами может быть отлична от нуля.

Div[{**1**, **1**, **1**}, {$\boldsymbol{r},\boldsymbol{\theta},\boldsymbol{\phi}$}, "**Spherical**"]//**Simplify**

$$\frac{2+\text{Cot}[\theta]}{r}$$

Функцию `Div` может применяться к спискам функций.

Div[{{**f11**[$\boldsymbol{x},\boldsymbol{y}$], **f12**[$\boldsymbol{x},\boldsymbol{y}$]}, {**f21**[$\boldsymbol{x},\boldsymbol{y}$], **f22**[$\boldsymbol{x},\boldsymbol{y}$]}}, {$\boldsymbol{x},\boldsymbol{y}$}]

$\{\text{f12}^{(0,1)}[x,y] + \text{f11}^{(1,0)}[x,y], \text{f22}^{(0,1)}[x,y] + \text{f21}^{(1,0)}[x,y]\}$

Ротор. Функция `Curl[{f1,f2,f3},{x1,x2,x3}]` вычисляет ротор вектор – функции $\left(f_1(x_1,x_2,x_3),\ f_2(x_1,x_2,x_3),\ f_3(x_1,x_2,x_3)\right)$ по формуле

$$\left(\frac{\partial f_3}{\partial x_2}-\frac{\partial f_2}{\partial x_3}, \frac{\partial f_1}{\partial x_3}-\frac{\partial f_3}{\partial x_1}, \frac{\partial f_2}{\partial x_1}-\frac{\partial f_1}{\partial x_2}\right).$$

Curl[{$\boldsymbol{y},-\boldsymbol{x},\boldsymbol{2z}$}, {$\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}$}]

$\{0, 0, -2\}$

Для вектор – функции двух переменных команда `Curl[{f1,f2},{x1,x2}]` вычисляет $\frac{\partial f_2}{\partial x_1} - \frac{\partial f_1}{\partial x_2}$.

$\mathbf{Curl}[\{\boldsymbol{y^2}, \boldsymbol{x^3}\}, \{\boldsymbol{x}, \boldsymbol{y}\}]$

$3x^2 - 2y$

Функция `Curl` работает даже тогда, когда ее первым аргументом является скалярная функция.

$\mathbf{Curl}[\boldsymbol{f}[\boldsymbol{x}], \{\boldsymbol{x}\}]$

$f'[x]$

$\mathbf{Curl}[\boldsymbol{f}[\boldsymbol{x}, \boldsymbol{y}], \{\boldsymbol{x}, \boldsymbol{y}\}]$

$\{-f^{(0,1)}[x, y], f^{(1,0)}[x, y]\}$

Ротор можно вычислять в криволинейных системах координат. Для этого имеется следующий формат вызова

`Curl[{f1,...,fn},{x1,...,xn},"система"],`

где третьим аргументом передается строка – имя системы координат.

$\mathbf{Curl}[\{\boldsymbol{r}\mathbf{Sin}[\boldsymbol{2\theta}], -\boldsymbol{r}\mathbf{Cos}[\boldsymbol{2\theta}]\}, \{\boldsymbol{r}, \boldsymbol{\theta}\}, \text{"}\mathbf{Polar}\text{"}]$

$-4\mathrm{Cos}[2\theta]$

В криволинейной системе координат ротор вектор – функции с постоянными компонентами может быть отличен от нуля.

$\mathbf{Curl}[\{\mathbf{1}, \mathbf{1}, \mathbf{1}\}, \{\boldsymbol{r}, \boldsymbol{\theta}, \boldsymbol{\phi}\}, \text{"}\mathbf{Spherical}\text{"}]$

$\left\{\frac{\mathrm{Cot}[\theta]}{r}, -\frac{1}{r}, \frac{1}{r}\right\}$

Ротор градиента равен нулю. Действительно

$\mathbf{Curl}[\mathbf{Grad}[\boldsymbol{f}[\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}], \{\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}\}], \{\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}\}]$

`{0,0,0}`

Векторное поле, ротор которого тождественно равен нулю, называется безвихревым. Результат последней команды показывает, что векторное поле, являющееся градиентом некоторого скалярного поля, является безвихревым. Скалярное поле, градиентом которого является векторное поле, называется потенциалом этого векторного поля.

Например, потенциал поля точечного заряда q в пространстве задается выражением $u = -\frac{k\,q}{r}$, где r расстояние от точки до заряда, а k – некоторая постоянная. Действительно, градиент скалярного поля u порождает векторное поле $\boldsymbol{F}$

$\boldsymbol{u}[\mathbf{r\_}, \boldsymbol{\varphi}\mathbf{\_}, \boldsymbol{\theta}\mathbf{\_}] = -\frac{\boldsymbol{kq}}{\boldsymbol{r}};$

$\boldsymbol{F} = \mathbf{Grad}[\boldsymbol{u}[\boldsymbol{r}, \boldsymbol{\varphi}, \boldsymbol{\theta}], \{\boldsymbol{r}, \boldsymbol{\varphi}, \boldsymbol{\theta}\}, \text{"}\mathbf{Spherical}\text{"}]$

$\{\frac{kq}{r^2}, 0, 0\}$

Поскольку сила притяжения направлена вдоль радиуса – вектора точки и обратно пропорционально квадрату расстояния, то последнее выражение показывает, что векторное поле $\boldsymbol{F}$ является полем силы притяжения. Кроме того

Curl[F,{r,φ,θ},"**Spherical**"]

```
{0,0,0}
```

Легко видеть, что любое векторное поле, задаваемое в сферической системе координат в виде $(f(r),0,0)$, является безвихревым.

Curl[{f[r],**0**,**0**},{r,θ,ϕ},"**Spherical**"]

```
{0,0,0}
```

Иногда функция `Curl` возвращает разреженную матрицу, которая выглядит не совсем обычно

s = **Curl**[$x+y+z$,{x,y,z}]

```
StructuredArray[SymmetrizedArray,{3,3},-Structured Data-]
```

Чтобы преобразовать результат в обычную матрицу к нему нужно применить функцию `Normal`.

Normal[s]

$\{\{0,1,-1\},\{-1,0,1\},\{1,-1,0\}\}$

Между дифференциальными операторами векторного анализа имеются связи. Например, $grad\,(u\cdot v)=u\cdot grad\,(v)+v\cdot grad\,(u)$. Это тождество легко проверить. Имеем

Remove[**u**,**v**]

Grad[u[x,y,z] v[x,y,z],{x,y,z}] == u[x,y,z] **Grad**[v[x,y,z],{x,y,z}] + v[x,y,z] **Grad**[u[x,y,z],{x,y,z}]

```
True
```

Для скалярного $h(x,y,z)$ и векторного поля $\mathbf{A}(x,y,z)$ выполняются тождества $div(h\cdot\mathbf{A})=h\,div\,\mathbf{A}+\langle\mathbf{A},grad\,h\rangle$ и $rot(h\cdot\mathbf{A})=h\,rot\,\mathbf{A}-[\mathbf{A},grad\,h]$, где угловые и квадратные скобки обозначают скалярное и векторное произведение. Проверим их. Для первого тождества имеем

Clear[u,v,w,h];

A = {u[x,y,z],v[x,y,z],w[x,y,z]};

Div[h[x,y,z]A,{x,y,z}] == h[x,y,z]**Div**[A,{x,y,z}] + A.**Grad**[h[x,y,z],{x,y,z}]//**Simplify**

```
True
```

Проверим второе тождество.

Curl[h[x,y,z] A,{x,y,z}] == h[x,y,z] **Curl**[A,{x,y,z}] − A × **Grad**[h[x,y,z],{x,y,z}]//**Simplify**

```
True
```

Напомним, что векторное произведение (функция `Cross`) в виде × (крест) вводится как `Esc – cross – Esc`.

Имеются и другие тождества, связывающие дифференциальные операции первого порядка. Вы можете проверить их самостоятельно.

Дифференциальные операции 2 – го порядка. Вычисляя градиент скалярного поля, мы производим над ним дифференциальную операцию первого порядка. Аналогично можно говорить о дифференциальных операциях первого порядка над векторным полем (операции взятия дивергенции и ротора). Каждая операция первого порядка приводит к новому полю – векторному или

скалярному. Если над ними произвести снова операцию первого порядка, то над исходным полем будет произведена операция второго порядка.

Пусть исходное поле является скалярным (функция u). Над ним можно произвести только одну дифференциальную операцию *grad u*. Новое поле является векторным и над ним можно произвести одну из двух операций: взять дивергенцию или ротор. Таким образом, над скалярным полем можно произвести две операции второго порядка: *rot grad u* и *div grad u*. Для первой операции имеем

```
Curl[Grad[u[x, y, z], {x, y, z}], {x, y, z}]
{0, 0, 0}
```

Векторное поле, ротор которого равен нулю, называется безвихревым. Сейчас мы показали что векторное поле, которое образует градиент произвольной скалярной функции, является безвихревым.

Для второй операции в декартовой системе координат имеем

```
Div[Grad[u[x, y, z], {x, y, z}], {x, y, z}]
```

$$u^{(0,0,2)}[x,y,z]+u^{(0,2,0)}[x,y,z]+u^{(2,0,0)}[x,y,z]$$

Операция 2 – го порядка *div grad u* приводит к новому скалярному полю. Она называется «оператором Лапласа» или «лапласианом» и обозначается коротко Δu. Для вычисления этой операции в *Mathematica* имеется специальная функция `Laplacian`.

```
Laplacian[u[x, y, z], {x, y, z}]
```

$$u^{(0,0,2)}[x,y,z]+u^{(0,2,0)}[x,y,z]+u^{(2,0,0)}[x,y,z]$$

```
Laplacian[Sqrt[x^2 + y^2 + z^2], {x, y, z}]//Simplify
```

$$\frac{2}{\sqrt{x^2+y^2+z^2}}$$

```
Laplacian[f[x, y], {x, y}]
```

$$f^{(0,2)}[x,y]+f^{(2,0)}[x,y]$$

Скалярное поле, лапласиан которого в некоторой области тождественно равен нулю, называется гармоническим (гармонической функцией).

$$u[x,y,z]=\frac{1}{\sqrt{x^2+y^2+z^2}};$$

```
Laplacian[u[x, y, z], {x, y, z}]//Simplify
0
```

Т.е. функция $u=\frac{1}{\sqrt{x^2+y^2+z^2}}$ является гармонической везде, кроме точки (0,0,0), в которой она не определена.

Первым аргументом функции `Laplacian`. может быть список функций или массив большей размерности. В декартовой системе лапласиан от списка функций равен списку лапласианов этих функций.

```
Laplacian[{x^2 + y^2, Sqrt[x^2 + y^2], 1/Sqrt[x^2 + y^2], Log[x^2 + y^2]}, {x, y}]//Simplify
```

$$\{4, \frac{1}{\sqrt{x^2+y^2}}, \frac{1}{(x^2+y^2)^{3/2}}, 0\}$$

Однако в криволинейных системах координат это не так. Лапласиан даже постоянного вектора может иметь ненулевые координаты

```
Laplacian[{1, 1}, {r, φ}, "Polar"]//Simplify
```

$$\left\{-\frac{1}{r^2}, -\frac{1}{r^2}\right\}$$

Обратите внимание, что в двумерном пространстве функция $1/\sqrt{x^2+y^2}$ не является гармонической. Гармонической является функция $\ln\left(x^2+y^2\right)$.

Рассмотрим теперь дифференциальные операции 2 – го порядка над векторным полем. Операций первого порядка две: *div* **A** и *rot* **A**. Над скалярным полем *div* **A** можно произвести только операцию взятия градиента.

```
Clear[u, v, w];
A = {u[x, y, z], v[x, y, z], w[x, y, z]};
gd = Grad[Div[A, {x, y, z}], {x, y, z}]
```

$$\{w^{(1,0,1)}[x,y,z] + v^{(1,1,0)}[x,y,z] + u^{(2,0,0)}[x,y,z],$$
$$w^{(0,1,1)}[x,y,z] + v^{(0,2,0)}[x,y,z] + u^{(1,1,0)}[x,y,z],$$
$$w^{(0,0,2)}[x,y,z] + v^{(0,1,1)}[x,y,z] + u^{(1,0,1)}[x,y,z]\}$$

Над векторным полем *rot* **A** можно произвести операции *div* и *rot*; это приводит нас еще к двум операциям второго порядка: *div rot* **A** и *rot rot* **A**. Но первая операция приводит к тождественному нулю

```
Div[Curl[A, {x, y, z}], {x, y, z}]
```

0

Вторая операция *rot rot* **A** дает

```
cc = Curl[Curl[A, {x, y, z}], {x, y, z}]
```

$$\{-u^{(0,0,2)}[x,y,z] - u^{(0,2,0)}[x,y,z] + w^{(1,0,1)}[x,y,z] + v^{(1,1,0)}[x,y,z],$$
$$-v^{(0,0,2)}[x,y,z] + w^{(0,1,1)}[x,y,z] + u^{(1,1,0)}[x,y,z] - v^{(2,0,0)}[x,y,z],$$
$$v^{(0,1,1)}[x,y,z] - w^{(0,2,0)}[x,y,z] + u^{(1,0,1)}[x,y,z] - w^{(2,0,0)}[x,y,z]\}$$

Можно заметить, что между операциями *rot rot* **A** и *grad div* **A** существует связь: *grad div* $\mathbf{A} = \Delta\mathbf{A} + $ *rot rot* **A** (в декартовой системе координат). Действительно

```
gd == Laplacian[A, {x, y, z}] + cc
```

```
True
```

Мы уже говорили, что функция `Curl` умеет работать со скалярными функциями и результатом является вектор.

```
Curl[u[x, y], {x, y}]
```

$$\{-u^{(0,1)}[x,y], u^{(1,0)}[x,y]\}$$

Ее повторное использование возвращает Лапласиан, т.е. двойной ротор скалярного поля является Лапласианом этого поля. Действительно,

```
Curl[Curl[f[x, y], {x, y}], {x, y}]
```

$$f^{(0,2)}[x,y] + f^{(2,0)}[x,y]$$

Аналогичный результат имеет место для пространственного скалярного поля.

Curl[Curl[$f[x,y,z]$,{x,y,z}],{x,y,z}]

$f^{(0,0,2)}[x,y,z]+f^{(0,2,0)}[x,y,z]+f^{(2,0,0)}[x,y,z]$

Выше мы показали, что дивергенция ротора векторного поля тождественно равна нулю. Но определение функции `Curl` таково, что и для скалярного поля дивергенция ротора равна нулю.

Div[Curl[$f[x,y,z]$,{x,y,z}],{x,y,z}]//Normal

```
{0,0,0}
```

Div[Curl[$f[x,y]$,{x,y}],{x,y}]

0

С другими нестандартными возможностями функций `Grad`, `Div`, `Curl` вы можете познакомиться по справочной системе. В частности, в нашем пособии мы не рассмотрели примеры того, как работают эти функции с тензорными полями.

Особую роль в приложениях играют **потенциальные поля**. Векторное поле $\vec{\mathbf{A}}=\left(A_x, A_y, A_z\right)$ называется потенциальным, если существует скалярная функция *u*, для которой $\vec{\mathbf{A}}$ служит градиентом $\vec{\mathbf{A}}=grad\,u$. Сама функция *u* называется потенциальной функцией (или потенциалом) поля $\vec{\mathbf{A}}$. Для того, чтобы поле $\vec{\mathbf{A}}$ было потенциальным необходимо и достаточно, чтобы во всей рассматриваемой области $rot\,\vec{\mathbf{A}}$ обращался в нуль.

Векторное поле, ротор которого тождественно равен нулю, называется безвихревым. Таким образом, понятия потенциального и безвихревого поля совпадают.

В курсах математического анализа доказывается, что для такого поля циркуляция по простому замкнутому контуру всегда будет нулем, а линейный интеграл по кривой, соединяющей любые две точки поля, оказывается не зависящим от формы кривой. Сама потенциальная функция *u*, с точностью до произвольного постоянного слагаемого, определяется криволинейным интегралом 2 – го рода

$$u=\int_C\left\langle\vec{\mathbf{A}},d\,\mathbf{s}\right\rangle=\int_C\vec{\mathbf{A}}\cdot d\,\mathbf{s}=\int_C A_x\,d\,x+A_y\,d\,y+A_z\,d\,z\,,$$

взятым от некоторой фиксированной точки Q до переменной точки P рассматриваемой области по любой соединяющей эти точки кривой C. Т.о., если точки P и Q являются концевыми точками кривой C, то для потенциального поля имеет место

$$\int_C\vec{\mathbf{A}}\cdot d\,\mathbf{s}=u(P)-u(Q)$$

Потенциальную функцию *u* векторного поля можно находить по – другому. Например, пусть имеется плоское векторное поле $\vec{\mathbf{A}}=\left(A_x, A_y\right)$. Если оно

потенциальное, то $\vec{\mathbf{A}} = grad\, u = \left(\frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}\right) = \left(A_x, A_y\right)$. Т.е. $\frac{\partial u}{\partial x} = A_x$ и $\frac{\partial u}{\partial y} = A_y$.

Решив эту систему относительно $u(x,y)$, вы найдете потенциальную функцию.

Пример. Показать, что векторное поле $\mathbf{F} = \left(3x^2 - 2xy + 2, 6y^2 - x^2 + 3\right)$ является безвихревым и найти его потенциал.

```
Clear[F1, F2, F, x, y]
F1[x_, y_] = 3x^2 - 2xy + 2;
F2[x_, y_] = 6y^2 - x^2 + 3;
F[x_, y_] = {F1[x, y], F2[x, y]};
Curl[F[x, y], {x, y}]
```

0

Поскольку $rot\,\mathbf{F} = 0$, то поле безвихревое и, следовательно, имеет потенциал. Для него имеем $grad\, u = \left(\frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}\right) = \left(F_1, F_2\right) = \left(3x^2 - 2xy + 2, 6y^2 - x^2 + 3\right)$. Тогда

```
u = ∫ F1[x, y] dx + h[y]
```

$2x + x^3 - x^2 y + h[y]$

Добавление $h(y)$ здесь необходимо, поскольку постоянная интегрирования может зависеть от y. Далее

```
s = Solve[D[u, y] == F2[x, y], h'[y]]
```

$\{\{h'[y] \to 3(1 + 2y^2)\}\}$

```
h[y_] = Integrate[h'[y] /. s[[1]], y]
```

$3y + 2y^3$

Таким образом, потенциал поля равен (печатаем $\boldsymbol{u}$, а подстановка значения $h(y)$ выполняется автоматически)

```
u
```

$2x + x^3 + 3y - x^2 y + 2y^3$

□

Пример. Показать, что векторное поле $\mathbf{F} = \left(3x^2 + 2xy, x^2 + 2y + z, y + 3z^2\right)$ является безвихревым и определить его потенциал.

```
Remove[x, y, z, F1, F2, F3, F, h, g]
F1[x_, y_, z_] = 3x^2 + 2xy;
F2[x_, y_, z_] = x^2 + 2y + z;
F3[x_, y_, z_] = y + 3z^2;
F = {F1[x, y, z], F2[x, y, z], F3[x, y, z]}
Curl[F, {x, y, z}]
```

{0, 0, 0}

Поскольку $rot\,\mathbf{F} = 0$, то поле безвихревое и, следовательно, имеет потенциал.

```
u = ∫ F1[x, y, z] dx + h[y, z]
```

$x^3 + x^2 y + h[y, z]$

Добавление $h(y,z)$здесь необходимо, поскольку постоянная интегрирования может зависеть от *y* и *z*. Далее

$\mathbf{s1 = Solve[}\boldsymbol{D[u,y]}\mathbf{ == F2[}\boldsymbol{x,y,z}\mathbf{],}\ \boldsymbol{\partial_y h[y,z]}\mathbf{]}$

$\{\{h^{(1,0)}[y,z] \to 2y + z\}\}$

$\boldsymbol{h}\mathbf{[y\_,z\_] = \int s1[[1,1,2]]}\,\boldsymbol{dy + g[z]}$

$y^2 + yz + g[z]$

Проверим, чему сейчас равно *u*.

$\boldsymbol{u}$

$x^3 + x^2y + y^2 + yz + g[z]$

$\mathbf{s2 = Solve[}\boldsymbol{D[u,z]}\mathbf{ == F3[}\boldsymbol{x,y,z}\mathbf{],}\ \boldsymbol{\partial_z g[z]}\mathbf{]}$

$\{\{g'[z] \to 3z^2\}\}$

$\boldsymbol{g}\mathbf{[z\_] = \int s2[[1,1,2]]}\,\boldsymbol{dz}$

z^3

Теперь

$\boldsymbol{u}$

$x^3 + x^2y + y^2 + yz + z^3$

Итак, потенциалом поля $\boldsymbol{F}$ является функция $u(x,y,z) = x^3 + x^2y + y^2 + yz + z^3$ и любая функция, отличная от *u* на константу.

Вычислим потенциал, используй криволинейный интеграл 2 – го рода. Потенциал u в точке (X,Y,Z) вычисляется по формуле

$$u(X,Y,Z) = \int_{(x_0,y_0,z_0)}^{(X,Y,Z)} \left(3x^2 + 2xy\right)dx + \left(x^2 + 2y + z\right)dy + \left(y + 3z^2\right)dz,$$

где в качестве начальной точки можно взять начало координат, а в качестве пути (интеграл не зависит от пути интегрирования) – любую линию, соединяющую точки $(0,0,0)$ и (X,Y,Z), например ломаную, изображенную на следующем рисунке.

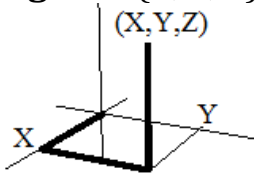
$\boldsymbol{X = 1; Y = 1; Z = 1;}$

$\mathbf{ParametricPlot3D[\{\{}\boldsymbol{tX}\mathbf{,0,0\},\{}\boldsymbol{X,tY}\mathbf{,0\},\{}\boldsymbol{X,Y,tZ}\mathbf{\}\},\{}\boldsymbol{t}\mathbf{,0,1\},}$

$\mathbf{PlotRange \to \{\{-0.5,1.5\},\{-0.5,1.5\},\{-0.5,1.5\}\},}$

$\mathbf{PlotStyle \to \{\{Black, Thickness[0.02]\}\},}$

$\mathbf{Boxed \to False, AxesOrigin \to \{0,0,0\}]}$



Интеграл по такой ломаной равен сумме интегралов по отрезкам. Имеем

$\mathbf{i1 = Integrate[F1[}\boldsymbol{x,y,z}\mathbf{],\{}\boldsymbol{x}\mathbf{,0,}\boldsymbol{X}\mathbf{\}]/.\{}\boldsymbol{y \to 0, z \to 0}\mathbf{\}}$

$\mathbf{i2 = Integrate[F2[}\boldsymbol{x,y,z}\mathbf{],\{}\boldsymbol{y}\mathbf{,0,}\boldsymbol{Y}\mathbf{\}]/.\{}\boldsymbol{x \to X, z \to 0}\mathbf{\}}$

$\mathbf{i3 = Integrate[F3[}\boldsymbol{x,y,z}\mathbf{],\{}\boldsymbol{z}\mathbf{,0,}\boldsymbol{Z}\mathbf{\}]/.\{}\boldsymbol{x \to X, y \to Y}\mathbf{\}}$

$\mathbf{i1 + i2 + i3}$

$X^3 + X^2Y + Y^2 + YZ + Z^3$

Мы пришли к тому же результату, что и выше.

□

Векторное поле $\vec{\mathbf{A}}$ называется **соленоидальным**, или трубчатым, если существует векторное поле $\vec{\mathbf{B}}$ для которого $\vec{\mathbf{A}}$ служит вихрем, т.е. $\vec{\mathbf{A}} = rot\,\vec{\mathbf{B}}$. Сам вектор $\vec{\mathbf{B}}$ называют векторным потенциалом поля $\vec{\mathbf{A}}$. Для того чтобы поле $\vec{\mathbf{A}}$ было соленоидальным, необходимо и достаточно, чтобы во всей рассматриваемой области выполнялось равенство $div\,\vec{\mathbf{A}} = 0$. Например,

```
A[x_, y_, z_] = {x y^2, x^2 y, -(x^2 + y^2) z};
Div[A[x, y, z], {x, y, z}]
0
```

и векторное поле $\boldsymbol{F}$ является соленоидальным.

Проверим, что следующее плоское векторное поле является соленоидальным.

```
A[x_, y_] = {x^2 y + y^3, x^3 - x y^2};
Div[A[x, y], {x, y}]
0
```

3.4.3 Криволинейные и поверхностные интегралы

Криволинейный интеграл 1 – го рода. Пусть в пространстве задано скалярное поле $u(x,y,z)$ и непрерывная кусочно – гладкая кривая C: $\mathbf{r}(t) = x(t)\mathbf{i} + y(t)\mathbf{j} + z(t)\mathbf{k}$. Криволинейным интегралом первого рода от функции $u(x,y,z)$ по кривой C называется число, равное

$$\int_C u(x,y,z)\,dl = \int_0^T u(x(t),y(t),z(t))\sqrt{x'(t)^2 + y(t)^2 + z(t)^2}\,dt \qquad (1)$$

или в векторной записи

$$\int_C u(x,y,z)\,ds = \int_0^T u(\mathbf{r}(t))\,\|\,\mathbf{r}'(t)\,\|\,dt,$$

где $\|\ \|$ обозначает норму/длину вектора.

Левая часть (1) есть обозначение интеграла первого рода, а правая есть способ его вычисления – это обычный определенный интеграл по t на [0,T]. Изначальное определение интеграла первого рода дается в терминах предела интегральных сумм. Интеграл на плоскости определяется аналогично.

Пример. Пусть вдоль винтовой линии $\mathbf{r}(t) = a\,(\mathbf{i}\cos t + \mathbf{j}\sin t) + bt\,\mathbf{k}$ распределена масса с линейной плотностью $u(x,y,z) = z^2$. Найти массу участка винтовой линии $0 \le t \le T$.

Масса кривой определяется криволинейным интегралом первого рода (1).

```
Clear[u, r, p]
r[t_] = {a Cos[t], a Sin[t], b t};
sn = Simplify[Norm[r'[t]] /. Abs[x_]^2 → x^2]
p = {x, y, z};
```

$\boldsymbol{u}[\mathbf{x\_}, \mathbf{y\_}, \mathbf{z\_}] = \boldsymbol{z}^{\mathbf{2}}$;
$\boldsymbol{M} = \mathbf{Integrate}[(\boldsymbol{u}@@\boldsymbol{p}/.\mathbf{Thread}[\mathbf{Rule}[\boldsymbol{p}, \boldsymbol{r}[\boldsymbol{t}]]]) * \mathbf{sn}, \{\boldsymbol{t}, \mathbf{0}, \boldsymbol{T}\}]$
$\frac{1}{3}b^2\sqrt{a^2+b^2}T^3$ (*ответ – масса кривой *)

Поясним некоторые элементы приведенного кода. Команда `Apply` (инфиксная форма @@) заменяет заголовок переменной $\boldsymbol{p}$ (это `List`) на u.

$\boldsymbol{u}@@\boldsymbol{p}$
$u[x, y, z]$

Команда `Thread[f[args]]` «проносит» функцию `f` сквозь любые списки, которые появляется в ее аргументах `args`. Например,

$\mathbf{Thread}[\boldsymbol{f}[\{\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}\}, \{\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}\}]]$
$\{f[a, x], f[b, y], f[c, z]\}$

В нашем примере функция `Thread` строит список подстановок

$\mathbf{Thread}[\mathbf{Rule}[\boldsymbol{p}, \boldsymbol{r}[\boldsymbol{t}]]]$
$\{x \to a\mathrm{Cos}[t], y \to a\mathrm{Sin}[t], z \to bt\}$

Это можно было записать еще так $\mathbf{Thread}[\boldsymbol{p} -> r[t]]$. В результате команда

$\boldsymbol{u}@@\boldsymbol{p}/.\mathbf{Thread}[\mathbf{Rule}[\boldsymbol{p}, \boldsymbol{r}[\boldsymbol{t}]]]$
$u[a\mathrm{Cos}[t], a\mathrm{Sin}[t], bt]$

выполняет подстановку параметрических уравнений $\boldsymbol{r}[t]$ кривой в аргументы функции $u[x,y,z]$, которая требуется в интеграле (1).

Функция `Norm[r'[t]]` вычисляет норму производной радиуса – вектора $\boldsymbol{r}[t]$, и запоминает ее в переменной **sn**. Подстановка `Abs[x_]^2 → x^2` любые выражения вида $Abs[expr]^2$ заменяет выражением $expr^2$ при любом $expr$.

□

Пример. Вычислить интеграл первого рода по эллипсу $x = a\cos t$, $y = b\sin t$ $(0 \le t \le 2\pi)$ от функции $u(x, y) = \sqrt{\left(\frac{b\,x}{a}\right)^2 + \left(\frac{a\,y}{b}\right)^2}$. Имеем

$\mathbf{Clear}[\boldsymbol{u}, \boldsymbol{r}, \boldsymbol{p}]$
$\boldsymbol{r}[\mathbf{t\_}] = \{\boldsymbol{a}\mathbf{Cos}[\boldsymbol{t}], \boldsymbol{b}\mathbf{Sin}[\boldsymbol{t}]\}$;
$\mathbf{sn} = \mathbf{Simplify}[\mathbf{Norm}[\boldsymbol{r}'[\boldsymbol{t}]]/.\mathbf{Abs}[\mathbf{x\_}]\mathbf{\hat{}2} \to \boldsymbol{x}\mathbf{\hat{}2}]$
$\boldsymbol{p} = \{\boldsymbol{x}, \boldsymbol{y}\}$;
$\boldsymbol{u}[\boldsymbol{x\_}, \boldsymbol{y\_}] = \sqrt{\left(\frac{\boldsymbol{b}\,\boldsymbol{x}}{\boldsymbol{a}}\right)^{\mathbf{2}} + \left(\frac{\boldsymbol{a}\,\boldsymbol{y}}{\boldsymbol{b}}\right)^{\mathbf{2}}}$;
$\boldsymbol{L} = \mathbf{Integrate}[(\boldsymbol{u}@@\boldsymbol{p}/.\mathbf{Thread}[\mathbf{Rule}[\boldsymbol{p}, \boldsymbol{r}[\boldsymbol{t}]]]) * \mathbf{sn}, \{\boldsymbol{t}, \mathbf{0}, \mathbf{2}\boldsymbol{\pi}\}]$
$(a^2 + b^2)\pi$

Криволинейный интеграл 2 – го рода. Пусть в пространстве задано векторное поле $\vec{\mathbf{F}}(x, y, z) = P(x, y, z)\mathbf{i} + Q(x, y, z)\mathbf{j} + R(x, y, z)\mathbf{k}$, где P, Q, R – скалярные функции, и непрерывная кусочно – гладкая кривая C: $\mathbf{r}(t) = x(t)\mathbf{i} + y(t)\mathbf{j} + z(t)\mathbf{k}$. Криволинейным интегралом второго рода от вектор – функции $\vec{\mathbf{F}}(x, y, z)$ по кривой C называется число, равное

$$\int_C \langle \mathbf{F}(x,y,z),\, d\,\boldsymbol{l}\rangle = \int_C \mathbf{F}\cdot d\,\boldsymbol{l} = \int_C P\,d\,x + Q\,d\,y + R\,d\,z =$$
$$= \int_0^T \left[\, P(x(t),y(t),z(t))\,x'(t) + Q(x(t),y(t),z(t))\,y'(t) + R(x(t),y(t),z(t))\,z'(t)\,\right] d\,t \qquad (2)$$

или в векторной записи

$$\int_C \mathbf{F}\cdot d\,\boldsymbol{l} = \int_C \mathbf{F}(x(t),y(t),z(t))\cdot \mathbf{r}'(t)\,d\,t \qquad (3)$$

Верхние записи в (2) представляют различные варианты обозначения интеграла второго рода, а нижняя представляет способ его вычисления – определенный интеграл по t в пределах [0,T]. Изначальное определение интеграла второго рода дается в терминах предела интегральных сумм. Интеграл для плоских кривых определяется аналогично.

Пример. Вычислить интеграл второго рода $I = \int_{AB} x^2 d\,x + x\,y\,d\,y$ вдоль прямолинейного отрезка , идущего из точки A(0, 0) в точку B(1, 1), и по дуге параболы $y = x^2$, соединяющей те же точки.

В первом случае $y=x$ имеем

```
Clear[F, F1, F2, p, x, y];
r1[x_] = {x, x};
p = {x, y};
F = {#1^2&, 1 * #2&};
tf = Table[F[[i]]@@p, {i, 2}] /. Thread[p → r1[x]]
Integrate[tf.r1'[x], {x, 0, 1}]
```

$$\frac{2}{3}$$

Во втором случае $y = x^2$.

```
r2[x_] = {x, x^2};
tf = Table[F[[i]]@@p, {i, 2}] /. Thread[p → r2[x]]
Integrate[tf.r2'[x], {x, 0, 1}]
```

$$\frac{11}{15}$$

Пример. Найти работу силового поля $\vec{\mathbf{F}} = x\,\mathbf{i} + y\,\mathbf{j} + z\,\mathbf{k}$ при перемещении материальной точки вдоль первого витка конической винтовой линии $x = a\,e^t \cos t$, $y = a\,e^t \sin t$, $z = a\,e^t$ из точки A(0,0,0) в точку B(a,0,a).

Заметим, что точке A соответствует значение параметра $t = -\infty$, а точке B соответствует $t = 0$. Имеем

```
Clear[F, r, p];
r[t_] = {aExp[t]Cos[t], aExp[t]Sin[t], aExp[t]};
p = {x, y, z};
F = {#1&, #2&, #3&};
tf = Table[F[[i]]@@p, {i, 3}] /. Thread[p → r[t]]
g[t_] = tf.r'[t]//Simplify
```

$\mathbf{A = Integrate[}\boldsymbol{g}\mathbf{[}\boldsymbol{t}\mathbf{], \{}\boldsymbol{t}\mathbf{, -}\infty\mathbf{, 0\}]}$
$\{a e^t \mathrm{Cos}[t], a e^t \mathrm{Sin}[t], a e^t\}$
$2a^2 e^{2t}$
a^2
Для наглядности мы привели результаты работы последних трех строк кода. Окончательно, работа $A = a^2$. □

Криволинейный интеграл второго рода от вектора $\boldsymbol{F}$, взятый по замкнутому контуру C, называется циркуляцией вектора поля по данному контуру и обозначается символом $\oint_C \vec{\mathbf{F}}\, d\boldsymbol{l}$. Направление обхода контура указывается заранее, причем положительным считается обход против часовой стрелки.

Для плоских векторных полей $\vec{\mathbf{F}} = (P(x,y), Q(x,y))$ выполняется формула Грина

$$\oint_C P\,dx + Q\,dy = \iint_D \left(\frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y}\right) dx\,dy, \qquad (4)$$

где D представляет область, ограниченную кривой C. Формула (4) справедлива только в случае, когда функции P и Q непрерывны вместе со своими производными $\frac{\partial Q}{\partial x}, \frac{\partial P}{\partial y}$ в замкнутой области $\overline{D} = D \bigcup C$.

Пример. С помощью формулы Грина вычислить криволинейный интеграл $\oint_C (x+y)dx - (x-y)dy$, где C – окружность $x^2 + y^2 = R^2$. Имеем

$\mathbf{Clear[}\boldsymbol{F}\mathbf{, }\boldsymbol{x}\mathbf{, }\boldsymbol{y}\mathbf{]}$
$\boldsymbol{F}\mathbf{[x\_, y\_] = \{}\boldsymbol{x} + \boldsymbol{y}\mathbf{, -(}\boldsymbol{x} - \boldsymbol{y}\mathbf{)\};}$
$\mathbf{Integrate[Curl[}\boldsymbol{F}\mathbf{[}\boldsymbol{x}\mathbf{, }\boldsymbol{y}\mathbf{], \{}\boldsymbol{x}\mathbf{, }\boldsymbol{y}\mathbf{\}] * Boole[}\boldsymbol{x}^2 + \boldsymbol{y}^2 \leq \boldsymbol{R}^2\mathbf{],}$
$\mathbf{\{}\boldsymbol{x}\mathbf{, -}\boldsymbol{R}\mathbf{, }\boldsymbol{R}\mathbf{\}, \{}\boldsymbol{y}\mathbf{, -}\boldsymbol{R}\mathbf{, }\boldsymbol{R}\mathbf{\}, Assumptions :\to }\boldsymbol{R} > 0\mathbf{]}$
$-2\pi R^2$

Пример. Используя формулу Грина (4), вычислить криволинейный интеграл $\oint_C e^{2x+y}\,dx + e^{-y}\,dy$ по контуру квадрата C с вершинами (0,0), (1,0), (1,1), (0,1).

Поскольку функции P и Q непрерывны вместе со своими производными $\frac{\partial Q}{\partial x}, \frac{\partial P}{\partial y}$ в области квадрата, то формула Грина применима. Имеем

$\boldsymbol{F}\mathbf{[x\_, y\_] = \{Exp[2}\boldsymbol{x} + \boldsymbol{y}\mathbf{], Exp[-}\boldsymbol{y}\mathbf{]\};}$
$\mathbf{Integrate[Curl[}\boldsymbol{F}\mathbf{[}\boldsymbol{x}\mathbf{, }\boldsymbol{y}\mathbf{], \{}\boldsymbol{x}\mathbf{, }\boldsymbol{y}\mathbf{\}], \{}\boldsymbol{x}\mathbf{, 0, 1\}, \{}\boldsymbol{y}\mathbf{, 0, 1\}]}$
$-\frac{1}{2}(-1 + e)^2(1 + e)$
$\boldsymbol{N}\mathbf{[\%]}$
-5.4890995

Поверхностный интеграл 1 – го рода. Пусть в пространстве задано скалярное поле $u(x,y,z)$ и гладкая поверхность S: $\mathbf{r}(u,v) = x(u,v)\mathbf{i} + y(u,v)\mathbf{j} + z(u,v)\mathbf{k}$, $|\mathbf{r}'_u \times \mathbf{r}'_v| > 0$, $(u,v) \in \Omega$, где Ω – ограниченная область с кусочно – гладкой

границей и $x(u,v), y(u,v), z(u,v)$ – непрерывно дифференцируемые на $\overline{\Omega}$ функции. Поверхностным интегралом первого рода называется число, равное

$$\iint\limits_S u(x,y,z)\,dS = \iint\limits_\Omega u(x(u,v),y(u,v),z(u,v))\,|\,\mathbf{r}'_u \times \mathbf{r}'_v\,|\,du\,dv, \qquad (5)$$

где справа стоит двойной интеграл по $(u,v)\in\Omega$. Левая часть (5) есть обозначение поверхностного интеграла первого рода, а правая дает способ его вычисления. Конечно, изначальное определение интеграла первого рода дается в терминах предела интегральных сумм.

В частности из (5) имеем следующую формулу для вычисления площади поверхности, заданной в параметрическом виде

$$S = \iint\limits_S dS = \iint\limits_\Omega |\,\mathbf{r}'_u \times \mathbf{r}'_v\,|\,du\,dv \qquad (6)$$

Пример. Используя формулу (6), вычислить площадь сферы радиуса R. Для этого уравнение сферы запишем в параметрическом виде $x = R\cos u \sin v$, $y = R\sin u \sin v$, $z = R\cos v$, где $0 \le u \le 2\pi$ и $0 \le v \le \pi$. Имеем

```
Clear[R, u, v, r1, r2]
r[u_, v_] = {RCos[u]Sin[v], RSin[u]Sin[v], RCos[v]};
r1[u_, v_] = D[r[u, v], u];
r2[u_, v_] = D[r[u, v], v];
nrm = Simplify[Norm[Cross[r1[u, v], r2[u, v]]] /. Abs[x_]^2 → x^2,
                                    Assumptions :→ R > 0]
S = Integrate[nrm, {u, 0, 2π}, {v, 0, π}]
```

$R^2\sqrt{\mathrm{Sin}[v]^2}$

$4\pi R^2$

Напомним, что подстановка Abs[x_]^2 → x^2 любые выражения вида $Abs[expr]^2$ заменяет выражением $expr^2$ при любом $expr$.

Пример. Вычислить интеграл $\iint_S x\,y\,z\,dS$ по поверхности $\mathbf{r}(u,v) = (u\cos v, u\sin v, v)$ при $0 \le u \le 1, 0 \le v \le 2\pi$.

```
Clear[R, u, v, r1, r2]
r[u_, v_] = {uCos[v], uSin[v], v};
r1[u_, v_] = D[r[u, v], u];
r2[u_, v_] = D[r[u, v], v];
n[u_, v_] = Simplify[Cross[r1[u, v], r2[u, v]]]
```

$\{\mathrm{Sin}[v], -\mathrm{Cos}[v], u\}$

```
nrm = Simplify[Norm[n[u, v]] /. Abs[x_]^2 -> x^2]
```

$\sqrt{1+u^2}$

```
p = {x, y, z};
u[x_, y_, z_] = x y z;
S = Integrate[(u@@p /. Thread[p -> r[u, v]]) * nrm, {v, 0, 2π}, {u, 0, 1}]
```

$\frac{1}{16}\pi(-3\sqrt{2} + \mathrm{ArcSinh}[1])$

$N[\%]$
```
-0.65998325
```

Пример. Вычислить массу поверхности конуса $z=\sqrt{x^2+y^2}$, ограниченной сверху плоскостью $z=h$, если поверхностная плотность пропорциональна расстоянию от этой точки до начала координат, т.е. $\rho(x,y,z)=k\sqrt{x^2+y^2+z^2}$.

Масса конической поверхности может быть вычислена по формуле $m=\iint\limits_S \rho(x,y,z)\,dS=\iint\limits_S k\sqrt{x^2+y^2+z^2}\,dS$. Тогда имеем

```
Clear[R, x, y, r1, r2]
r[x_, y_] = {x, y, Sqrt[x^2 + y^2]};
r1[x_, y_] = D[r[x, y], x];
r2[x_, y_] = D[r[x, y], y];
n[x_, y_] = Simplify[Cross[r1[x, y], r2[x, y]]];
nrm = Simplify[Norm[n[x, y]] /. Abs[x_]^2 -> x^2]
```
$\sqrt{2}$
```
p = {x, y, z};
u[x_, y_, z_] = k Sqrt[x^2 + y^2 + z^2];
m = Integrate[(u @@ p /. Thread[p -> r[x, y]]) * nrm * Boole[x^2 + y^2 <= h^2],
    {x, -h, h}, {y, -h, h}, Assumptions :> h > 0]
```
$\frac{4}{3}h^3k\pi$

Поверхностный интеграл 2 – го рода. Потоком вектора $\vec{\mathbf{F}}=P\mathbf{i}+Q\mathbf{j}+R\mathbf{k}$ через ориентированную поверхность **S** называется величина, обозначаемая $\int\limits_S \langle \mathbf{F}, d\mathbf{S}\rangle$ (или $\int\limits_S \mathbf{F}\cdot d\mathbf{S}$) и определяемая при помощи равенства

$$\int\limits_S \langle \mathbf{F}, d\mathbf{S}\rangle = \int\limits_S \langle \mathbf{F}, \mathbf{n}\rangle dS = \int\limits_S \mathbf{F}\cdot\mathbf{n}\,dS, \qquad (7)$$

в правой части которого стоят поверхностные интегралы первого рода от скалярного произведения $\langle \mathbf{F}, \mathbf{n}\rangle = P\cos(\mathbf{n},\mathbf{i})+Q\cos(\mathbf{n},\mathbf{j})+R\cos(\mathbf{n},\mathbf{k})$ вектора $\vec{\mathbf{F}}$ и единичной нормали **n**, определяющей ориентацию ***S***. Выражение в левой части (7) еще называют поверхностным интегралом второго рода. Изначальное определение интеграла второго рода дается в терминах предела интегральных сумм.

Если поверхность задана параметрически

$$\mathbf{r}(u,v)=x(u,v)\mathbf{i}+y(u,v)\mathbf{j}+z(u,v)\mathbf{k},\ \ |\mathbf{r}'_u\times\mathbf{r}'_v|>0,\ (u,v)\in\Omega,$$

где Ω – ограниченная область с кусочно – гладкой границей и $x(u,v), y(u,v), z(u,v)$ – непрерывно дифференцируемые на $\overline{\Omega}$ функции, то справедливо равенство

$$\int\limits_S \mathbf{F}\cdot\mathbf{n}\,dS=\int\limits_\Omega \mathbf{F}(x(u,v),y(u,v),z(u,v))\cdot\mathbf{n}(u,v)\,du\,dv, \qquad (8)$$

где в правой части стоит двойной интеграл по области $(u,v)\in\Omega$.

Пример. Вычислить интеграл $\int_S \mathbf{F}\cdot d\,\mathbf{S}$, где $\vec{\mathbf{F}}=x\,z\,\mathbf{i}+z\,\mathbf{j}+y\,x\,\mathbf{k}$ и $\mathbf{r}(u,v)=(u-v^2)\mathbf{i}+u\,v\,\mathbf{j}+(u^2-v)\mathbf{k}$, $0\le u\le 2$ и $1\le v\le 3$.

```
Clear[F, r, r1, r2, n]
r[u_, v_] = {u - v^2, u v, u^2 - v};
r1[u_, v_] = D[r[u, v], u];
r2[u_, v_] = D[r[u, v], v];
n[u_, v_] = Cross[r1[u, v], r2[u, v]]
p = {x, y, z};
F = {(#1#3)&, #3&, (#2#1)&};
tf = Table[F[[i]]@@p, {i, 3}] /. Thread[p → r[u, v]]
flux = Integrate[tf.n[u, v], {u, 0, 2}, {v, 1, 3}]
```

$$-\frac{6928}{15}$$

Здесь переменная **tf** содержит результат подстановки координат вектора $\boldsymbol{r}$ в выражение вектор функции, т.е. $\mathbf{F}(x(u,v),y(u,v),z(u,v))$.

Пример. Найти поток вектора $\vec{\mathbf{F}}=x\,\mathbf{i}+y\,\mathbf{j}+z\,\mathbf{k}$ через часть поверхности эллипсоида $\frac{x^2}{a^2}+\frac{y^2}{b^2}+\frac{z^2}{c^2}=1$, лежащую в первом октанте, в направлении внешней нормали.

Используем уравнение эллипсоида в параметрическом виде. Имеем.

```
Clear[F, r, r1, r2, n]
r[u_, v_] = {a Cos[u] Cos[v], b Cos[u] Sin[v], c Sin[u]};
r1[u_, v_] = D[r[u, v], u];
r2[u_, v_] = D[r[u, v], v];
n[u_, v_] = Simplify[-Cross[r1[u, v], r2[u, v]]]
```

$\{b\ c\ \mathrm{Cos}[u]^2\mathrm{Cos}[v], a\ c\ \mathrm{Cos}[u]^2\mathrm{Sin}[v], a\ b\ \mathrm{Cos}[u]\mathrm{Sin}[u]\}$

Обратите внимание, что при вычислении вектора нормали перед векторным произведением мы поставили знак минус. Это нужно, чтобы вектор **n** был внешней нормалью.

```
p = {x, y, z};
F[x_, y_, z_] = {x, y, z};
flx = Simplify[(F[x, y, z] /. Thread[p → r[u, v]]).n[u, v]];
flux = Integrate[flx, {u, 0, π/2}, {v, 0, π/2}]
```

$$\frac{1}{2}a\ b\ c\ \pi$$

Пример. Найти поток ротора векторного поля $\vec{\mathbf{F}}=(x^2+y^2)\mathbf{i}+(x+z^2)\mathbf{j}$ через часть поверхности конуса $z^2=x^2+y^2$ $(1\le z\le 4)$ в направлении внешней нормали.

Вначале составим параметрическое уравнение этой части конуса. Оно имеет вид $\mathbf{r}(u,v)=u\cos v\,\mathbf{i}+u\sin v\,\mathbf{j}+u\,\mathbf{k}$, $1\le u\le 4$ и $0\le v\le 2\pi$. Тогда имеем

```
Clear[F, r, r1, r2, n]
```

$\boldsymbol{r}[\mathbf{u\_}, \mathbf{v\_}] = \{\boldsymbol{u}\mathbf{Cos}[\boldsymbol{v}], \boldsymbol{u}\mathbf{Sin}[\boldsymbol{v}], \boldsymbol{u}\}$;
$\mathbf{r1}[\mathbf{u\_}, \mathbf{v\_}] = \boldsymbol{D}[\boldsymbol{r}[\boldsymbol{u}, \boldsymbol{v}], \boldsymbol{u}]$;
$\mathbf{r2}[\mathbf{u\_}, \mathbf{v\_}] = \boldsymbol{D}[\boldsymbol{r}[\boldsymbol{u}, \boldsymbol{v}], \boldsymbol{v}]$;
$\boldsymbol{n}[\mathbf{u\_}, \mathbf{v\_}] = \mathbf{Simplify}[-\mathbf{Cross}[\mathbf{r1}[\boldsymbol{u}, \boldsymbol{v}], \mathbf{r2}[\boldsymbol{u}, \boldsymbol{v}]]]$
$\{u\mathrm{Cos}[v], u\mathrm{Sin}[v], -u\}$
$\boldsymbol{p} = \{\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}\}$;
$\boldsymbol{F}[\mathbf{x\_}, \mathbf{y\_}, \mathbf{z\_}] = \{\boldsymbol{x}^2 + \boldsymbol{y}^2, \boldsymbol{x} + \boldsymbol{z}^2, \mathbf{0}\}$;
$\mathbf{flx} = \mathbf{Simplify}[(\mathbf{Curl}[\boldsymbol{F}[\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}], \{\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}\}]/.\mathbf{Thread}[\boldsymbol{p} \to \boldsymbol{r}[\boldsymbol{u}, \boldsymbol{v}]]).\boldsymbol{n}[\boldsymbol{u}, \boldsymbol{v}]]$
$\mathbf{flux} = \mathbf{Integrate}[\mathbf{flx}, \{\boldsymbol{v}, \mathbf{0}, \mathbf{2}\boldsymbol{\pi}\}, \{\boldsymbol{u}, \mathbf{1}, \mathbf{4}\}]$
-15π

Мы уже говорили, что криволинейный интеграл второго рода от вектора $\boldsymbol{F}$, взятый по замкнутому контуру C, называется циркуляцией вектора поля по данному контуру и обозначается символом $\oint_C \vec{\mathbf{F}}\, d\,\boldsymbol{l}$. Если поле плоское, то для него выполняется формула Грина. Если же поле и контур расположены в пространстве, то справедлива формула Стокса

$$\int_C \mathbf{F} \cdot d\,\boldsymbol{l} = \int_S rot\,\mathbf{F} \cdot d\,\mathbf{S}, \tag{9}$$

где справа стоит поверхностный интеграл второго рода от ротора векторного поля $\boldsymbol{F}$, а слева – криволинейный интеграл второго рода по контуру C поверхности, ориентированному соответственно ориентации $\boldsymbol{S}$. Формула (9) выражает тот факт, что поток вектора $rot\,\vec{\mathbf{F}}$ через ориентируемую поверхность $\boldsymbol{S}$ равен циркуляции вектора $\boldsymbol{F}$ по контуру C этой поверхности.

Вспоминая формулу (8), имеем

$$\int_C \mathbf{F} \cdot d\,\boldsymbol{l} = \int_\Omega \big(rot\,\mathbf{F}\big)\big(x(u,v), y(u,v), z(u,v)\big) \cdot \mathbf{n}(u,v)\, d\,u\, d\,v \tag{10}$$

Пример. Вычислить интеграл $\int_C \mathbf{F} \cdot d\,\boldsymbol{l}$, где $\vec{\mathbf{F}} = x\,y\,z\,\mathbf{i} + \big(z + 3x - 3y\big)\mathbf{j} + y^2\,x\mathbf{k}$ и C является окружностью радиуса R в плоскости xy. Имеем.

$\mathbf{Clear}[\boldsymbol{F}, \boldsymbol{r}, \boldsymbol{R}]$
$\boldsymbol{r}[\mathbf{t\_}] = \{\boldsymbol{R}\mathbf{Cos}[\boldsymbol{t}], \boldsymbol{R}\mathbf{Sin}[\boldsymbol{t}], \mathbf{0}\}$; (* параметрическое уравнение окружности C *)
$\boldsymbol{p} = \{\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}\}$;
$\boldsymbol{F}[\mathbf{x\_}, \mathbf{y\_}, \mathbf{z\_}] = \{\boldsymbol{x}\,\boldsymbol{y}\,\boldsymbol{z}, \boldsymbol{z} + \mathbf{3}\boldsymbol{x} - \mathbf{3}\boldsymbol{y}, \boldsymbol{y}^2\boldsymbol{x}\}$;
$\mathbf{circ} = \mathbf{Integrate}[(\boldsymbol{F}[\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}]/.\mathbf{Thread}[\boldsymbol{p} \to \boldsymbol{r}[\boldsymbol{t}]]).\boldsymbol{r}'[\boldsymbol{t}], \{\boldsymbol{t}, \mathbf{0}, \mathbf{2}\boldsymbol{\pi}\}]$
$3\pi R^2$

Теперь применим формулу (10). Имеем

$\mathbf{Clear}[\boldsymbol{F}, \boldsymbol{r}, \mathbf{r1}, \mathbf{r2}, \boldsymbol{n}, \boldsymbol{R}]$
$\boldsymbol{r}[\mathbf{u\_}, \mathbf{v\_}] = \{\boldsymbol{R}\mathbf{Cos}[\boldsymbol{u}]\mathbf{Sin}[\boldsymbol{v}], \boldsymbol{R}\mathbf{Sin}[\boldsymbol{u}]\mathbf{Sin}[\boldsymbol{v}], \boldsymbol{R}\mathbf{Cos}[\boldsymbol{v}]\}$;
$\mathbf{r1}[\mathbf{u\_}, \mathbf{v\_}] = \mathbf{D}[\mathbf{r}[\mathbf{u}, \mathbf{v}], \mathbf{u}]$;
$\mathbf{r2}[\mathbf{u\_}, \mathbf{v\_}] = \boldsymbol{D}[\boldsymbol{r}[\boldsymbol{u}, \boldsymbol{v}], \boldsymbol{v}]$;
$\boldsymbol{n}[\mathbf{u\_}, \mathbf{v\_}] = \mathbf{Simplify}[-\mathbf{Cross}[\mathbf{r1}[\boldsymbol{u}, \boldsymbol{v}], \mathbf{r2}[\boldsymbol{u}, \boldsymbol{v}]]]$
$\{R^2\mathrm{Cos}[u]\mathrm{Sin}[v]^2, R^2\mathrm{Sin}[u]\mathrm{Sin}[v]^2, R^2\mathrm{Cos}[v]\mathrm{Sin}[v]\}$
$\boldsymbol{p} = \{\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}\}$;
$\boldsymbol{F}[\mathbf{x\_}, \mathbf{y\_}, \mathbf{z\_}] = \{\boldsymbol{x}\boldsymbol{y}\boldsymbol{z}, \boldsymbol{z} + \mathbf{3}\boldsymbol{x} - \mathbf{3}\boldsymbol{y}, \boldsymbol{y}^2\boldsymbol{x}\}$;

$\mathbf{flx = Simplify[(Curl[}\boldsymbol{F}\mathbf{[}\boldsymbol{x,y,z}\mathbf{],\{}\boldsymbol{x,y,z}\mathbf{\}]/.Thread[}\boldsymbol{p \to r}\mathbf{[}\boldsymbol{u,v}\mathbf{]]).}\boldsymbol{n}\mathbf{[}\boldsymbol{u,v}\mathbf{]];}$
$\mathbf{flux = Integrate[flx,\{}\boldsymbol{v}\mathbf{,0,}\boldsymbol{\pi}\mathbf{/2\},\{}\boldsymbol{u}\mathbf{,0,}\boldsymbol{2\pi}\mathbf{\}]}$
$3\pi R^2$

Пример. Для векторного поля $F = \left(e\,x^3 - 3\,x\,y + z^3, 2\,z^3 - x\,z^2 + y^4, 6\,y + 2\,z^3 x^2\right)$ и поверхности S, являющейся частью параболоида $z = x^2 + y^2$ $\left(z \le 9\right)$, проверить теорему Стокса.

Вначале вычислим циркуляцию как криволинейный интеграл второго рода. Кривой C, по которой вычисляется циркуляция, является окружность радиуса 3, расположенная в плоскости $z = 9$. Записываем ее параметрическое уравнение и используем формулу (3). Имеем

$\mathbf{Clear[}\boldsymbol{t,r,F}\mathbf{]}$
$\boldsymbol{p} = \{\boldsymbol{x,y,z}\};$
$\boldsymbol{r}\mathbf{[t\_]} = \{\mathbf{3Cos[}\boldsymbol{t}\mathbf{],3Sin[}\boldsymbol{t}\mathbf{],9}\};$
$\boldsymbol{F}\mathbf{[x\_,y\_,z\_]} = \{\boldsymbol{x^3 e} - \mathbf{3}\boldsymbol{xy} + \boldsymbol{z^3}, \mathbf{2}\boldsymbol{z^3} - \boldsymbol{xz^2} + \boldsymbol{y^4}, \mathbf{6}\boldsymbol{y} + \mathbf{2}\boldsymbol{z^3 x^2}\};$
$\mathbf{circ = Integrate[(}\boldsymbol{F}\mathbf{[}\boldsymbol{x,y,z}\mathbf{]/.Thread[}\boldsymbol{p \to r}\mathbf{[}\boldsymbol{t}\mathbf{]]).}\boldsymbol{r'}\mathbf{[}\boldsymbol{t}\mathbf{],\{}\boldsymbol{t}\mathbf{,0,}\boldsymbol{2\pi}\mathbf{\}]}$
$-729\,\pi$

В формуле Стокса (10) в качестве поверхности, натянутой на кривую C, используем часть поверхности параболоида $z = x^2 + y^2$ $\left(z \le 9\right)$. Имеем

$\mathbf{Clear[}\boldsymbol{r,x,y}\mathbf{]}$
$\boldsymbol{r}\mathbf{[x\_,y\_]} = \{\boldsymbol{x,y,x^2 + y^2}\};$
$\mathbf{r1[x\_,y\_]} = \boldsymbol{D}\mathbf{[}\boldsymbol{r}\mathbf{[}\boldsymbol{x,y}\mathbf{],}\boldsymbol{x}\mathbf{]};$
$\mathbf{r2[x\_,y\_]} = \boldsymbol{D}\mathbf{[}\boldsymbol{r}\mathbf{[}\boldsymbol{x,y}\mathbf{],}\boldsymbol{y}\mathbf{]};$
$\boldsymbol{n}\mathbf{[x\_,y\_]} = \mathbf{Simplify[Cross[r1[}\boldsymbol{x,y}\mathbf{],r2[}\boldsymbol{x,y}\mathbf{]]]}$
$\{-2x, -2y, 1\}$
$\mathbf{flx = Simplify[(Curl[}\boldsymbol{F}\mathbf{[}\boldsymbol{x,y,z}\mathbf{],\{}\boldsymbol{x,y,z}\mathbf{\}]/.}\boldsymbol{z \to x^2 + y^2}\mathbf{).}\boldsymbol{n}\mathbf{[}\boldsymbol{x,y}\mathbf{]];}$
$\mathbf{flux = Integrate[flx * Boole[}\boldsymbol{x^2 + y^2 \le 9}\mathbf{],\{}\boldsymbol{x}\mathbf{,-3,3\},\{}\boldsymbol{y}\mathbf{,-3,3\}]}$
-729π

Значения циркуляции, вычисленные разными способами, совпали.

Пусть V будет областью в $\mathbf{R}^3$ с границей ∂V, которая является кусочно – гладкой поверхностью, ориентированной так, что ее вектор нормали направлен наружу этой области. И пусть $\mathbf{F}\left(x, y, z\right)\big)$ является векторным полем с непрерывными частными производными в области $\overline{V}$. Тогда выполняется формула Гаусса – Остроградского

$$\iint\limits_{\partial V} \mathbf{F} \cdot d\,\mathbf{S} = \iiint\limits_{V} div\,\mathbf{F}\,d\,V \qquad (11)$$

Она говорит, что поток векторного поля $\boldsymbol{F}$ через замкнутую поверхность ∂V, лежащую в этом поле, в направлении ее внешней нормали, равен тройному (объемному) интегралу по области V, ограниченной этой поверхностью, от дивергенции этого поля.

Пример. Вычислить поток векторного поля $\vec{\mathbf{F}} = \left(x, y^2, y+z\right)$ через поверхность S тела, являющегося частью цилиндра $x^2 + y^2 \le 4$ расположенной между плоскостями $z = x$ и $z = 8$. Имеем

```
Clear[x, y, z, F]
F[x_, y_, z_] = {x, y^2, y + z};
Integrate[Div[F[x, y, z], {x, y, z}] * Boole[x^2 + y^2 <= 4 && x <= z && z <= 8],
                        {x, -2, 2}, {y, -2, 2}, {z, -∞, ∞}]
```

64π

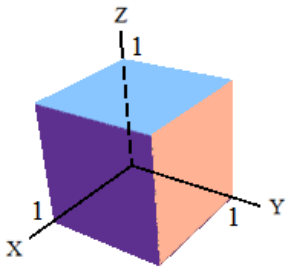
Пример. Используя теорему Гаусса – Остроградского, найти поток вектора $\mathbf{F} = x^3\,\mathbf{i} + y^3\,\mathbf{j} + R^2 z\,\mathbf{k}$ через всю поверхность тела $\frac{H}{R^2}\left(x^2 + y^2\right) \le z \le H$.

```
Clear[x, y, z, F, R, H]
F[x_, y_, z_] = {x^3, y^3, R^2 z};
div = Div[F[x, y, z], {x, y, z}]
Integrate[div * Boole[H/R^2 (x^2 + y^2) <= z <= H],
      {x, -∞, ∞}, {y, -∞, ∞}, {z, -∞, ∞}, Assumptions :→ H > 0 && R > 0]
```

$R^2 + 3x^2 + 3y^2$

$H\pi R^4$

Пример. Для векторного поля $\mathbf{F} = x\,y\,z\,\mathbf{i} + \left(x^2 + y^2 + z^2\right)\mathbf{j} + \left(x\,y + y\,z + x\,z\right)\mathbf{k}$ проверить формулу (11), если областью V является единичный куб, показанный на следующем рисунке.



Вначале вычисляем поток векторного поля через границу куба. Для этого вычислим поток поля через каждую его грань, учитывая направление внешней нормали. Имеем

```
Clear[x, y, z, F]
F[x_, y_, z_] = {xyz, x^2 + y^2 + z^2, xy + yz + xz};
nz = {0, 0, 1};
Iz0 = Integrate[-F[x, y, 0].nz, {x, 0, 1}, {y, 0, 1}]
Iz1 = Integrate[F[x, y, 1].nz, {x, 0, 1}, {y, 0, 1}]
```

$-\frac{1}{4}$

$\frac{5}{4}$

Здесь через **nz** мы обозначили единичный вектор, направленный вдоль оси Z, а через **Iz0** и **Iz1** потоки поля через грани куба z=0 и z=1. При этом в интеграле

Iz0 мы учли, что вектор нормали направлен в сторону, противоположную вектору **nz**. Аналогичные обозначения мы используем ниже для других граней.

$\mathbf{nx} = \{\mathbf{1}, \mathbf{0}, \mathbf{0}\};$

$\mathbf{Ix0} = \mathbf{Integrate}[-\boldsymbol{F}[\mathbf{0}, \boldsymbol{y}, \boldsymbol{z}].\,\mathbf{nx}, \{\boldsymbol{y}, \mathbf{0}, \mathbf{1}\}, \{\boldsymbol{z}, \mathbf{0}, \mathbf{1}\}]$

$\mathbf{Ix1} = \mathbf{Integrate}[\boldsymbol{F}[\mathbf{1}, \boldsymbol{y}, \boldsymbol{z}].\,\mathbf{nx}, \{\boldsymbol{y}, \mathbf{0}, \mathbf{1}\}, \{\boldsymbol{z}, \mathbf{0}, \mathbf{1}\}]$

$\mathbf{ny} = \{\mathbf{0}, \mathbf{1}, \mathbf{0}\};$

$\mathbf{Iy0} = \mathbf{Integrate}[-\boldsymbol{F}[\boldsymbol{x}, \mathbf{0}, \boldsymbol{z}].\,\mathbf{ny}, \{\boldsymbol{x}, \mathbf{0}, \mathbf{1}\}, \{\boldsymbol{z}, \mathbf{0}, \mathbf{1}\}]$

$\mathbf{Iy1} = \mathbf{Integrate}[\boldsymbol{F}[\boldsymbol{x}, \mathbf{1}, \boldsymbol{z}].\,\mathbf{ny}, \{\boldsymbol{x}, \mathbf{0}, \mathbf{1}\}, \{\boldsymbol{z}, \mathbf{0}, \mathbf{1}\}]$

$\mathbf{flux1} = \mathbf{Iz0} + \mathbf{Iz1} + \mathbf{Ix0} + \mathbf{Ix1} + \mathbf{Iy0} + \mathbf{Iy1}$

$\frac{9}{4}$

Теперь используем формулу (11). Имеем

$\mathbf{flux2} = \mathbf{Integrate}[\mathbf{Div}[\boldsymbol{F}[\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}], \{\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}\}], \{\boldsymbol{x}, \mathbf{0}, \mathbf{1}\}, \{\boldsymbol{y}, \mathbf{0}, \mathbf{1}\}, \{\boldsymbol{z}, \mathbf{0}, \mathbf{1}\}]$

$\frac{9}{4}$

Значения совпали, но вычисления вторым способом значительно короче.

3.4.4 Криволинейные системы координат

В системе *Mathematica* имеется несколько функций, предназначенных для преобразования математических понятий из одной системы координат (СК) в другую.

Формулы перехода из одной системы координат в другую возвращает функция `CoordinateTransform`.

Функция `CoordinateTransformData` вычисляет специфические понятия, связанные с переходом из одной СК в другую. Это формулы перехода, матрица Якоби, якобиан (определитель матрицы Якоби), обратная матрица Якоби и некоторые другие.

Функция `TransformedField` выполняет преобразование скалярных и векторных полей при переходе из одной системы координат в другую.

Функция `CoordinateChartData` возвращает величины, характеризующие систему координат, например, метрические коэффициенты.

Команда `CoordinateTransform[преобразование,точка]` возвращает координаты точки в СК, заданной в «преобразовании». Например,

$\{\boldsymbol{x}, \boldsymbol{y}\} = \mathbf{CoordinateTransform}[\text{"}\mathbf{Polar}\text{"} \rightarrow \text{"}\mathbf{Cartesian}\text{"}, \{\boldsymbol{r}, \boldsymbol{\varphi}\}]$

$\{r\mathrm{Cos}[\varphi], r\mathrm{Sin}[\varphi]\}$

Здесь первый аргумент `"Polar"→"Cartesian"` определяет имена СК (в нашем примере выполняется переход из полярной системы в декартову); второй аргумент `{r,φ}` задает координаты точки в исходной СК. Имена результирующих координат вы можете задавать произвольно в левой части. В нашем случае мы определили, что переменные в декартовой/результирующей системе координат будут иметь имена *x* и *y*.

Второй аргумент может иметь конкретные/числовые значения координат; также он может быть списком координат точек.

CoordinateTransform[{"Cartesian"−> "Polar"}, {{**1**, **0**}, {**1**, **1**}, {**0**, **1**}}]

$\{\{1,0\},\{\sqrt{2},\frac{\pi}{4}\},\{1,\frac{\pi}{2}\}\}$

CoordinateTransform["Spherical" → "Cartesian", {**1**, $\boldsymbol{\pi}$/**4**, $\boldsymbol{\pi}$/**2**}]

$\{0,\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}}\}$

Пусть, например, кривая задана параметрическими уравнениями в полярной системе координат. Используя функцию `CoordinateTransform`, можно получить параметрические уравнения кривой в декартовых координатах, а затем построить ее график.

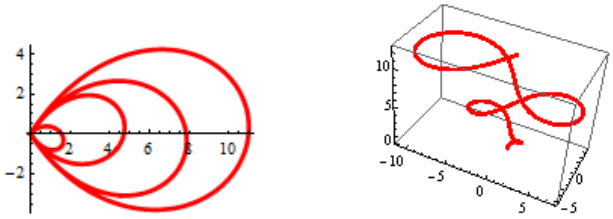
plr[t_]:= {$\boldsymbol{t}$ **Sin**[$\boldsymbol{t}$]^**2**, **Cos**[$\boldsymbol{t}$]};

cart[t_] = **CoordinateTransform**["Polar" → "Cartesian", **plr**[$\boldsymbol{t}$]]

{tCos[Cos[t]]Sin[t]^2, tSin[t]^{2}Sin[Cos[t]]}

ParametricPlot[**cart**[$\boldsymbol{t}$], {$\boldsymbol{t}$, **0**, **4**$\boldsymbol{\pi}$}, **PlotStyle** → {**Red**, **Thickness**[**0.01**]}]

(следующий рисунок слева).



Вот аналогичный пример кривой, заданной параметрически в цилиндрической системе координат

cyl[t_]:= {$\boldsymbol{t}$**Sin**[$\boldsymbol{t}$], **Cos**[$\boldsymbol{t}$], $\boldsymbol{t}$};

cart[t_] = **CoordinateTransform**["Cylindrical" → "Cartesian", **cyl**[$\boldsymbol{t}$]]

{tCos[Cos[t]]Sin[t], tSin[t]Sin[Cos[t]], t}

ParametricPlot3D[**cart**[$\boldsymbol{t}$], {$\boldsymbol{t}$, **0**, **4**$\boldsymbol{\pi}$}, **PlotStyle** → {**Red**, **Thickness**[**0.01**]}]

(предыдущий рисунок справа).

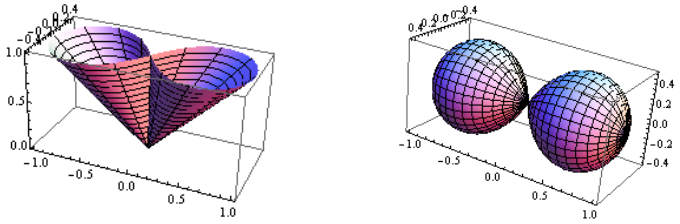
В цилиндрической (или другой допустимой) системе координат вы можете задать параметрическое уравнение поверхности, преобразовать его к декартовым координатам и построить график поверхности.

cyl[u_, v_]:= {$\boldsymbol{u}$**Sin**[$\boldsymbol{v}$], **Cos**[$\boldsymbol{v}$], $\boldsymbol{u}$};

cart[u_, v_] = **CoordinateTransform**["Cylindrical" → "Cartesian", **cyl**[$\boldsymbol{u}$, $\boldsymbol{v}$]]

{uCos[Cos[v]]Sin[v], uSin[v]Sin[Cos[v]], u}

ParametricPlot3D[**cart**[$\boldsymbol{u}$, $\boldsymbol{v}$], {$\boldsymbol{u}$, **0**, **1**}, {$\boldsymbol{v}$, **0**, **2**$\boldsymbol{\pi}$}] (* следующий рис. слева *)



Аналогично, зададим поверхность в параметрическими уравнениями в сферической системе координат и построим ее, перейдя к параметрическим уравнениям в декартовой системе.

sph[u_, v_]:= {**Sin**[$\boldsymbol{v}$], **Cos**[$\boldsymbol{v}$], $\boldsymbol{u}$};

cart[u_, v_] = **CoordinateTransform**["Spherical" → "Cartesian", **sph**[$\boldsymbol{u}$, $\boldsymbol{v}$]]

ParametricPlot3D[**cart**[u, v], {$u, 0, \pi$}, {$v, 0, 2\pi$}, **Mesh** → {**15**, **60**}]
(предыдущий рисунок справа).

Функция `CoordinateTransformData` вычисляет значения величин, связанных с переходом из одной СК в другую. Это могут быть формулы перехода, матрица Якоби, якобиан (определитель матрицы Якоби), обратная матрица Якоби и некоторые другие.

Для получения формул перехода из одной системы координат в другую можно использовать функцию `CoordinateTransformData`

{x, y} = **CoordinateTransformData**["**Polar**" → "**Cartesian**",
"**Mapping**", {r, φ}]

{rCos[φ], rSin[φ]}

Здесь первый аргумент `"Polar"→"Cartesian"` определяет имена систем координат; второй аргумент определяет, что функция будет возвращать (`Mapping` говорит, что функция вернет формулы перехода); третий аргумент `{r,φ}` задает имена отображаемых переменных (в нашем случае имена переменных в полярной системе координат). Имена результирующих координат можно задавать в левой части.

CoordinateTransformData["**Cartesian**" → "**Spherical**", "**Mapping**", {x, y, z}]

$\{\sqrt{x^2+y^2+z^2}, \text{ArcTan}[z, \sqrt{x^2+y^2}], \text{ArcTan}[x, y]\}$

Если третий аргумент не передавать, то будут возвращены «чистые» функции.

CoordinateTransformData["**Polar**" → "**Cartesian**", "**Mapping**"]

{Cos[#1[[2]]] #1[[1]], Sin[#1[[2]]] #1[[1]]}&

Если вы хотите получить матрицу Якоби замены переменных, то второй аргумент должен называться `"MappingJacobian"`

MJ = **CoordinateTransformData**["**Polar**" → "**Cartesian**",
"**MappingJacobian**", {r, φ}];

MJ//MatrixForm

$$\begin{pmatrix} \text{Cos}[\varphi] & -r\text{Sin}[\varphi] \\ \text{Sin}[\varphi] & r\text{Cos}[\varphi] \end{pmatrix}$$

Если вы хотите получить Якобиан (детерминант матрицы Якоби), то второй аргумент должен называться `"MappingJacobianDeterminant"`

Jac = **CoordinateTransformData**["**Polar**" → "**Cartesian**",
"**MappingJacobianDeterminant**", {r, φ}]

r

Второй аргумент может быть `"InverseMappingJacobian"`. В этом случае вы получите обратную матрицу Якоби.

CoordinateTransformData["**Polar**" → "**Cartesian**",
"**InverseMappingJacobian**", {r, φ}]//**MatrixForm**

$$\begin{pmatrix} \text{Cos}[\varphi] & \text{Sin}[\varphi] \\ -\dfrac{\text{Sin}[\varphi]}{r} & \dfrac{\text{Cos}[\varphi]}{r} \end{pmatrix}$$

Имена всех свойств, которые могут стоять вторым аргументом, можно получить командой

CoordinateTransformData["**Properties**"]

```
{InverseMappingJacobian,Mapping,
 MappingJacobian,MappingJacobianDeterminant,
 OrthonormalBasisRotation,StandardName}
```

Функция `TransformedField` умеет преобразовывать скалярные и векторные поля при переходе из одной системы координат в другую.

Clear$[x, y, r, \varphi]$

TransformedField["**Polar**" → "**Cartesian**", r^2**Cos**$[\theta]$, $\{r, \theta\} \to \{x, y\}$]

$x\sqrt{x^2 + y^2}$

TransformedField["**Cartesian**" → "**Polar**", $x^2 + y^2$, $\{x, y\} \to \{r, \varphi\}$]//**Simplify**

r^2

В этих двух примерах вторым аргументом стояло выражение (скалярное поле), которое надо было преобразовать к новой системе координат.

В следующем примере преобразуется векторное поле (т.е. вектор) (x, y).

TransformedField[{"**Cartesian**" → "**Polar**", **2**}, $\{x, y\}$, $\{x, y\} \to \{r, \varphi\}$]//**Simpl**ify

```
{r,0}
```

В полярной системе координат это поле определяется как $(r, 0)$.

Важно помнить, что имена переменных систем координат должны различаться. Например

Clear$[x, y, z, r, \varphi, z]$;

TransformedField["**Cartesian**" → "**Cylindrical**", $x^2 + y^2 + z^2$, $\{x, y, z\} \to \{r, \varphi, \zeta\}$]//**Simplify**

$r^2 + \zeta^2$

Здесь мы вынуждены вертикальную координату z цилиндрической системы назвать ζ, иначе будет сообщение об ошибке.

Вторым аргументом может стоять постоянный вектор

TransformedField["**Polar**" → "**Cartesian**", $\{1, 0\}$, $\{r, \varphi\}$−> $\{x, y\}$]

$\{\frac{x}{\sqrt{x^2 + y^2}}, \frac{y}{\sqrt{x^2 + y^2}}\}$

TransformedField["**Cartesian**"−> "**Polar**", $\{1, 1\}$, $\{x, y\}$−> $\{r, \varphi\}$]

$\{\text{Cos}[\varphi] + \text{Sin}[\varphi], \text{Cos}[\varphi] - \text{Sin}[\varphi]\}$

Пример. Потенциал поля тяготения (с точностью до постоянного множителя) задан в сферических координатах. Вычислить его градиент в декартовой СК.

u[r_, θ_, φ_] = **1**/r;

sphGrad = **Grad**[$u[r, \theta, \varphi]$, $\{r, \theta, \varphi\}$, "**Spherical**"]

$\{-\frac{1}{r^2}, 0, 0\}$

TransformedField["**Spherical**" → "**Cartesian**", **sphGrad**, $\{r, \theta, \varphi\} \to \{x, y, z\}$]

$\{-\frac{x}{(x^2 + y^2 + z^2)^{3/2}}, -\frac{y}{(x^2 + y^2 + z^2)^{3/2}}, -\frac{z}{(x^2 + y^2 + z^2)^{3/2}}\}$

Можно вначале преобразовать потенциал u к декартовым координатам, а потом вычислить его градиент.

```
U[x_, y_, z_] = TransformedField["Spherical" → "Cartesian",
                                 u[r, θ, φ], {r, θ, φ} → {x, y, z}]
```

$$\frac{1}{\sqrt{x^2+y^2+z^2}}$$

```
Grad[U[x, y, z], {x, y, z}, "Cartesian"]
```

$$\left\{-\frac{x}{(x^2+y^2+z^2)^{3/2}}, -\frac{y}{(x^2+y^2+z^2)^{3/2}}, -\frac{z}{(x^2+y^2+z^2)^{3/2}}\right\}$$

Пример. Потенциал поля задан в сферических координатах $u(r,\theta,\varphi)=\frac{p\cos\theta}{r^2}$ (электростатический потенциал диполя *p*, расположенного в начале координат и ориентированного вдоль оси z). Построить линии тока соответствующего векторного поля (силовые линии электрического поля) в плоскости YZ.

Векторное силовое поле вычисляется как градиент потенциала. Имеем

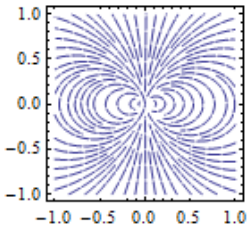
```
p = 1;  u[r_, θ_, φ_] = p Cos[θ] / r^2;
Esph = -Grad[u[r, θ, φ], {r, θ, φ}, "Spherical"]
```

$$\left\{\frac{2\text{Cos}[\theta]}{r^3}, \frac{\text{Sin}[\theta]}{r^3}, 0\right\}$$

```
Ecart = TransformedField["Spherical" → "Cartesian",
                         Esph, {r, θ, φ} → {x, y, z}]
```

$$\left\{\frac{3xz}{(x^2+y^2+z^2)^{5/2}}, \frac{3yz}{(x^2+y^2+z^2)^{5/2}}, -\frac{x^2+y^2}{(x^2+y^2+z^2)^{5/2}}+\frac{2z^2}{(x^2+y^2+z^2)^{5/2}}\right\}$$

```
StreamPlot[Rest[Ecart] /. x → 0, {y, -1, 1}, {z, -1, 1}]
```



Вначале мы вычислили силовое поле **Esph** в сферической СК. Затем преобразовали его к декартовым координатам. После этого отбросили первую координату векторного поля **Ecart** (функция `Rest` отбрасывает первый элемент списка), а в оставшиеся координатные функции подставили *x*=0.

Пример. Вычислить массу поверхности конуса $z=\sqrt{x^2+y^2}$, ограниченной сверху плоскостью *z=h*, если поверхностная плотность пропорциональна расстоянию от точки поверхности до начала координат, т.е. $\rho(x,y,z)=k\sqrt{x^2+y^2+z^2}$. Мы уже решали эту задачу в предыдущем пункте, выполняя все вычисления в декартовой СК.

Масса любой поверхности может быть вычислена по формуле $m=\iint\limits_S \rho(x,y,z)\,dS$. Если поверхность S задана параметрическими уравнениями

$\mathbf{r}(u,v)=x(u,v)\mathbf{i}+y(u,v)\mathbf{j}+z(u,v)\mathbf{k}$, $|\mathbf{r}'_u\times\mathbf{r}'_v|>0$, $(u,v)\in\Omega$, то поверхностный интеграл первого рода сводится к двойному интегралу по Ω. В нашем случае мы имеем

$$m=\iint_{\Omega} k\sqrt{x(u,v)^2+y(u,v)^2+z(u,v)^2}\,|\mathbf{r}'_u\times\mathbf{r}'_v|\,d\,u\,d\,v$$

Параметрическое уравнение конуса можем представить в виде $\mathbf{r}(x,y)=\left(x,y,\sqrt{x^2+y^2}\right)$. Вначале вычисляем подынтегральное выражение в декартовых координатах (в коде оно обозначено как $\boldsymbol{v}$).

```
Clear[R, x, y, r1, r2]
r[x_, y_] = {x, y, Sqrt[x^2 + y^2]};
r1[x_, y_] = D[r[x, y], x];
r2[x_, y_] = D[r[x, y], y];
n[x_, y_] = Simplify[Cross[r1[x, y], r2[x, y]]];
nrm = Simplify[Norm[n[x, y]] /. Abs[x_]^2 -> x^2]
p = {x, y, z};
u[x_, y_, z_] = k Sqrt[x^2 + y^2 + z^2];
v = Simplify[(u @@ p /. Thread[p -> r[x, y]]) * nrm]
```

$2k\sqrt{x^2+y^2}$

Теперь преобразуем $\boldsymbol{v}$ к полярным координатам

```
vpol = Simplify[TransformedField["Cartesian" → "Polar", v,
                {x, y} → {r, φ}], Assumptions :→ r > 0]
```

$2k\,r$

Вычисляем якобиан замены переменных

```
jac = CoordinateTransformData["Polar" → "Cartesian",
          "MappingJacobianDeterminant", {r, φ}]
```

```
r
```

Теперь вычисляем двойной интеграл по кругу $r\le h$ в полярных координатах

```
m = Integrate[vpol * jac, {φ, 0, 2π}, {r, 0, h}]
```

$\frac{4}{3}h^3k\pi$

Функция `CoordinateChartData` возвращает величины, характеризующие систему координат, например, метрические коэффициенты. В формате `CoordinateChartData["Сист. координат","свойство","точка"]` вычисляется значение «`свойства`» системы координат в `точке`. Например, метрические коэффициенты СК вычисляются командой

```
CoordinateChartData["Cartesian", "Metric", {x, y, z}]//MatrixForm
```

$$\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}$$

```
CoordinateChartData["Polar", "Metric", {r, φ}]//MatrixForm
```

$$\begin{pmatrix}1&0\\0&r^2\end{pmatrix}$$

CoordinateChartData["**Spherical**", "**Metric**", {r, θ, φ}]//**MatrixForm**

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & r^2 & 0 \\ 0 & 0 & r^2 \text{Sin}[\theta]^2 \end{pmatrix}$$

Список названий свойств (второй аргумент), которые можно узнать о СК, можно получить командой

CoordinateChartData["**Properties**"]

{… , Dimension, InverseMetric, Metric, … }

Вот примеры получения значений некоторых свойств СК.

CoordinateChartData["**Cylindrical**", "**Dimension**"]

3

CoordinateChartData["**Polar**", "**ScaleFactors**", {r, θ}]

$\{1, r\}$

CoordinateChartData["**Polar**", "**CoordinateRangeAssumptions**", {r, φ}]

$r > 0 \;\&\&\; -\pi < \varphi \leq \pi$

Если третий аргумент не задан, то обычно возвращается «чистая» функция.

test = CoordinateChartData["**Polar**", "**CoordinateRangeAssumptions**"]

#1[[1]]>0 && –π < #1[[2]] ≤ π &

Последнее свойство можно использовать для проверки того, правильно ли заданы координаты точки в соответствующей СК.

test[{**3**, π/**2**}]

True

test[{**3**, **3**π/**2**}]

False

test = CoordinateChartData[{"**Spherical**", **3**}, "**CoordinateRangeAssumptions**"]

#1[[1]]>0 && 0 < #1[[2]] < π && – π < #1[[3]] ≤ π &

test[{**3**, **3**π/**2**, **0**}]

False

Вычислим коэффициент пропорциональности между квадратом дифференциала $d\,s$ длины дуги кривой и квадратом дифференциала $d\,t$ параметра кривой ($d\,s^2 = \left|\mathbf{r}'(t)\right|^2 d\,t^2$) в декартовой СК.

$\boldsymbol{r}$[**t_**] = {$\boldsymbol{x}$[$\boldsymbol{t}$], $\boldsymbol{y}$[$\boldsymbol{t}$], $\boldsymbol{z}$[$\boldsymbol{t}$]}; (* параметрическое уравнение кривой *)

metric[**t_**] = **CoordinateChartData**[{"**Cartesian**", **3**}, "**Metric**"][$\boldsymbol{r}$[$\boldsymbol{t}$]]

{{1,0,0}, {0,1,0}, {0,0,1}}

$\boldsymbol{r}'$[$\boldsymbol{t}$].**metric**[$\boldsymbol{t}$].$\boldsymbol{r}'$[$\boldsymbol{t}$]

$x'[t]^2 + y'[t]^2 + z'[t]^2$

Вычислим аналогичный коэффициент пропорциональности для кривой $r = r(\varphi)$ в полярных координатах.

Clear[$\boldsymbol{r}$]

$\boldsymbol{R}$[$\boldsymbol{\varphi}$_] = {$\boldsymbol{r}$[$\boldsymbol{\varphi}$], $\boldsymbol{\varphi}$};

metric[$\boldsymbol{\varphi}$_] = **CoordinateChartData**[{"**Polar**", **2**}, "**Metric**"][$\boldsymbol{R}$[$\boldsymbol{\varphi}$]]

$\boldsymbol{R}'$[$\boldsymbol{\varphi}$].**metric**[$\boldsymbol{\varphi}$].$\boldsymbol{R}'$[$\boldsymbol{\varphi}$]

{{1,0}, {0, $r[\varphi]^2$}}

$r[\varphi]^2 + r'[\varphi]^2$

Т.е. мы получили, что $ds = \sqrt{r^2(\varphi) + r'^2(\varphi)}d\varphi$.

Параметр "VolumeFactor" позволяет вычислить числовой множитель для элемента объема. Например

$\boldsymbol{V = \text{CoordinateChartData}[\{\text{"Spherical"}, 3\}, \text{"VolumeFactor"}, \{r, \theta, \varphi\}]}$

$r^2 \text{Sin}[\theta]$

Зная его, можно вычислять объемы тел, границы которых задаются в сферической СК. Например, объем шара теперь можно вычислить по формуле

$$\int_0^R \int_0^{\pi} \int_0^{2\pi} V \, d\varphi \, d\theta \, dr$$

$$\frac{4\pi R^3}{3}$$

Проверим формулу площади круга. Имеем

$\boldsymbol{s = \text{CoordinateChartData}[\{\text{"Polar"}, 2\}, \text{"VolumeFactor"}, \{r, \varphi\}]}$

r

$$\int_0^R \int_0^{2\pi} s \, d\varphi \, dr$$

πR^2

Заметим, что иногда при указании системы координат следует задавать размерность пространства. Для этого вместо имени СК надо указать список, составленный из имени СК и размерности. Например, в последней команде полярную систему координат мы задали как {"Polar",2}.

Приведенные в этом пункте функции умеют работать с большим количеством систем координат. Читатель может познакомиться с ними самостоятельно по справочной системе.

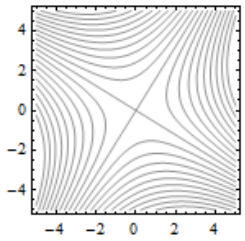
3.4.5 Графическое представление векторных полей.

Способы графического представления скалярных полей (скалярных функций двух и трех переменных) описаны нами ранее в главе «Графические возможности системы *Mathematica*». Напомним одну из таких функций.

$\boldsymbol{f[\text{x\_}, \text{y\_}] = x^2 + xy - y^2;}$

$\boldsymbol{\text{ContourPlot}[f[x, y], \{x, -5, 5\}, \{y, -5, 5\},}$

$\boldsymbol{\text{ContourShading} \to \text{False}, \ \text{Contours} \to 20]}$



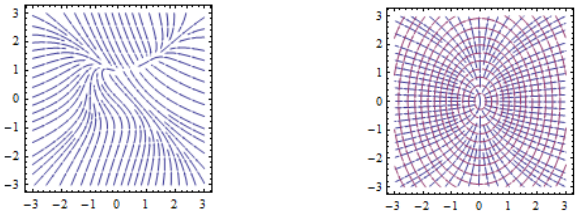
Здесь мы рассмотрим функции, которые выполняют визуализацию векторных полей (плоских и пространственных). Хотя они имеют дело с векторными

данными, но по своим возможностям, форматам использования и опциям оформления они близки к `ContourPlot`.

Для графического представления плоских векторных полей можно использовать функцию `StreamPlot`. Она строит линии тока (векторные линии, силовые линии) поля $\vec{\mathbf{F}} = (P(x,y), Q(x,y))$. Линии тока никогда не пересекаются, и через каждую точку проходит одна линия за исключением тех точек, где поле не определено или равно нулю. В декартовых координатах дифференциальные уравнения линий тока имеют вид $\frac{dx}{P(x,y)} = \frac{dy}{Q(x,y)}$.

В формате `StreamPlot[{P,Q},{x,`x_{min}`,`x_{max}`},{y`$_{min}$`,y`$_{max}$`}]` строится график линий тока векторного поля $(P(x,y), Q(x,y))$ на плоскости xy.

StreamPlot[{$-\mathbf{1} - \boldsymbol{x}^{\mathbf{3}} + \boldsymbol{y}^{\mathbf{2}}, \mathbf{1} + \boldsymbol{x}^{\mathbf{2}} - \boldsymbol{y}^{\mathbf{3}}$}, {$\boldsymbol{x}, -\mathbf{3}, \mathbf{3}$}, {$\boldsymbol{y}, -\mathbf{3}, \mathbf{3}$}]
(следующий рисунок слева)



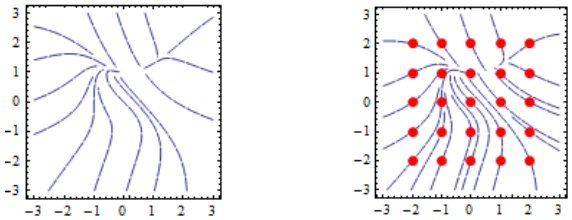
Можно строить линии тока нескольких векторных полей на одном графике.

StreamPlot[{{$\boldsymbol{x}, \boldsymbol{y}/\mathbf{2}$}, {$\boldsymbol{y}/\mathbf{2}, -\boldsymbol{x}$}}, {$\boldsymbol{x}, -\mathbf{3}, \mathbf{3}$}, {$\boldsymbol{y}, -\mathbf{3}, \mathbf{3}$}]
(предыдущий рисунок справа).

Опция `StreamPoints` позволяет указать количество линий тока, которое будет нарисовано или задать множество точек, через которые линии тока будет проходить.

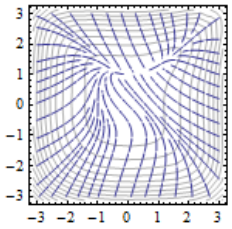
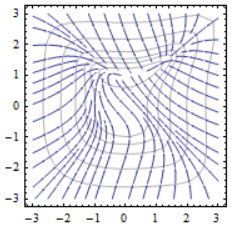
```
StreamPlot[{-1 - x^3 + y^2, 1 + x^2 - y^3}, {x, -3, 3}, {y, -3, 3},
        StreamPoints → 15]                    (*следующий рисунок слева *)
points = Flatten[Table[{x, y}, {x, -2, 2, 1}, {y, -2, 2, 1}], 1]

StreamPlot[{-1 - x^3 + y^2, 1 + x^2 - y^3}, {x, -3, 3}, {y, -3, 3},
        Epilog → {Red, PointSize[Large], Point[points]},
        StreamPoints → points]                (*следующий рисунок справа *)
```



Опция `Mesh->n` позволяет нарисовать `n` линий уровня нормы векторного поля вместе с линиями тока.

```
StreamPlot[{-1 - x^3 + y^2, 1 + x^2 - y^3}, {x, -3, 3}, {y, -3, 3},
        StreamPoints → 50, Mesh → 15]    (*следующий рисунок слева *)
```

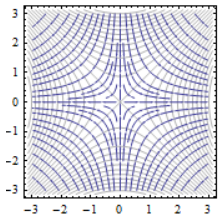
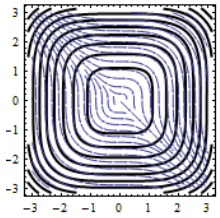
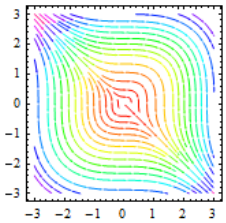
Можно задать конкретные значения этих линий уровня

$$\mathbf{StreamPlot}[\{-1-x^3+y^2, 1+x^2-y^3\}, \{x,-3,3\}, \{y,-3,3\},$$
$$\mathbf{StreamPoints} \to 50, \mathbf{Mesh} \to \{\{1,2,3,5,10,20\}\}]$$

(предыдущий рисунок справа).

По умолчанию строятся линии уровня величины/нормы векторного поля. Однако с помощью опции `MeshFunctions` можно определить скалярную функцию, линии уровня которой будут изображаться при использовании опции `Mesh`.

$$\mathbf{StreamPlot}[\{x,-y\}, \{x,-3,3\}, \{y,-3,3\}, \mathbf{Mesh} \to 25,$$
$$\mathbf{MeshFunctions} -> \mathbf{Function}[\{x,y,\mathbf{vx},\mathbf{vy},n\}, x^2-y^2]] \quad (* \text{ рисунок слева } *)$$

Опция MeshStyle управляет стилем этих линий уровня

$$\mathbf{StreamPlot}[\{y^2,-x^2\}, \{x,-3,3\}, \{y,-3,3\}, \mathbf{Mesh} \to 10,$$
$$\mathbf{MeshStyle} -> \{\mathbf{Black}, \mathbf{Thick}\}]$$

(предыдущий рисунок в середине).

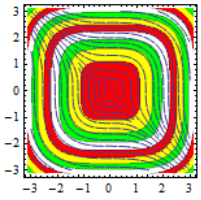
Величину/норму поля можно различать по цвету, используя опцию `StreamColorFunction`.

$$\mathbf{StreamPlot}[\{y^2,-x^2\}, \{x,-3,3\}, \{y,-3,3\}, \mathbf{StreamColorFunction} \to \mathbf{Hue}]$$

(предыдущий рисунок справа).

Области между линиями уровня можно закрашивать, используя опцию `MeshShading`.

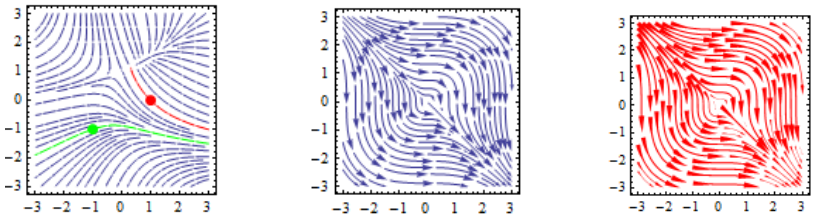
$$\mathbf{StreamPlot}[\{y^2,-x^2\}, \{x,-3,3\}, \{y,-3,3\}, \mathbf{Mesh} \to 10,$$
$$\mathbf{MeshShading} \to \{\mathbf{Red}, \mathbf{Yellow}, \mathbf{Green}, \mathbf{None}\}]$$



Можно управлять стилем линий тока – толщиной, цветом, размером стрелочек и т.д. Некоторые линии тока можно выделить, указав для них отдельный стиль, например, цвет (следующий рисунок слева).

$$\mathbf{StreamPlot}[\{-1-x\text{\^{}}2+y, 1+x-y\text{\^{}}2\}, \{x,-3,3\}, \{y,-3,3\},$$
$$\mathbf{StreamPoints} \to \{\{\{\{1,0\}, \mathbf{Red}\}, \{\{-1,-1\}, \mathbf{Green}\}, \mathbf{Automatic}\}\},$$

$$\mathbf{Epilog} \to \{\mathbf{Red}, \mathbf{PointSize}[\mathbf{Large}], \mathbf{Point}[\{\mathbf{1}, \mathbf{0}\}], \mathbf{Green}, \mathbf{Point}[\{-\mathbf{1}, -\mathbf{1}\}]\}]$$



Здесь для линий тока, которые проходят через точки (1,0) и (-1,-1), мы указали индивидуальный стиль (в нашем примере цвет), а для остальных линий – указали стиль Automatic.

Опция `StreamScale` управляет размером «наконечника» и длиной стрелок (предыдущий рисунок в середине).

$$\mathbf{StreamPlot}[\{\boldsymbol{y}^2, -\boldsymbol{x}^2\}, \{\boldsymbol{x}, -\mathbf{3}, \mathbf{3}\}, \{\boldsymbol{y}, -\mathbf{3}, \mathbf{3}\}, \mathbf{StreamScale} \to \mathbf{0.25}]$$

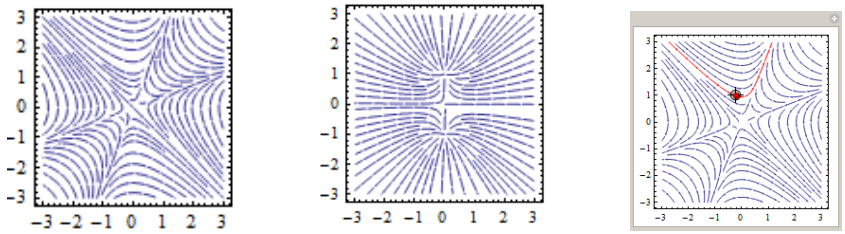
Опция `StreamStyle` управляет стилем линий тока

$$\mathbf{StreamPlot}[\{\boldsymbol{y}^2, -\boldsymbol{x}^2\}, \{\boldsymbol{x}, -\mathbf{3}, \mathbf{3}\}, \{\boldsymbol{y}, -\mathbf{3}, \mathbf{3}\}, \mathbf{StreamScale} \to \mathbf{0.4},$$
$$\mathbf{StreamStyle} \to \{\text{"}\mathbf{Dart}\text{"}, \mathbf{Red}\}] \quad \text{(* предыдущий рисунок справа *)}$$

Для предварительного символьного вычисления выражения векторного поля необходимо использовать функцию `Evaluate`.

$$\mathbf{StreamPlot}[\mathbf{Evaluate}[\mathbf{Grad}[\boldsymbol{y}^2\boldsymbol{x} - \boldsymbol{x}^2\boldsymbol{y}, \{\boldsymbol{x}, \boldsymbol{y}\}]], \{\boldsymbol{x}, -\mathbf{3}, \mathbf{3}\}, \{\boldsymbol{y}, -\mathbf{3}, \mathbf{3}\}]$$

(следующий рисунок слева). Попробуйте выполнить последнюю команду, опустив функцию `Evaluate`. Вы получите несколько сообщений и график построен не будет.



В качестве координат векторного поля можно использовать вещественные и мнимые части комплексных функций.

$$\boldsymbol{z} = \boldsymbol{x} + \boldsymbol{I}\boldsymbol{y};$$

$$\mathbf{StreamPlot}[\mathbf{Evaluate@}\{\mathbf{Re}[\boldsymbol{z} + \mathbf{1}/\boldsymbol{z}], \mathbf{Im}[\boldsymbol{z} + \mathbf{1}/\boldsymbol{z}]\}, \{\boldsymbol{x}, -\mathbf{3}, \mathbf{3}\}, \{\boldsymbol{y}, -\mathbf{3}, \mathbf{3}\}]$$

(предыдущий рисунок в середине).

Можно использовать динамические элементы системы для интерактивного слежения за линиями тока. Например,

$$\mathbf{Manipulate}[\ \mathbf{StreamPlot}[\ \mathbf{Evaluate}[\mathbf{Grad}[\boldsymbol{y}^2\boldsymbol{x} - \boldsymbol{x}^2\boldsymbol{y}, \{\boldsymbol{x}, \boldsymbol{y}\}]],$$
$$\{\boldsymbol{x}, -\mathbf{3}, \mathbf{3}\}, \{\boldsymbol{y}, -\mathbf{3}, \mathbf{3}\}, \mathbf{StreamPoints} \to \{\{\{\boldsymbol{p}, \mathbf{Red}\}, \mathbf{Automatic}\}\},$$
$$\mathbf{Epilog} \to \{\mathbf{Red}, \mathbf{PointSize}[\mathbf{Large}], \mathbf{Point}[\boldsymbol{p}]\}],$$
$$\{\{\boldsymbol{p}, \{\mathbf{1}, \mathbf{0}\}\}, \mathbf{Locator}\}] \quad \text{(* предыдущий рисунок справа *)}$$

«Захватите» мышью «локатор» и перемещайте его. Каждое новое положение «локатора» задает новую точку и, следовательно, новую линию тока. Кривая будет динамически перерисовываться.

Можно рисовать линии тока и векторное поле совместно. Для этого нужно использовать опцию `VectorPoints`. Векторное поле изображается множеством векторов – стрелочек на плоскости, направление которых

совпадает с направлением векторного поля, а длина в некотором масштабе равняется значению нормы векторного поля в соответствующих точках. Если `VectorPoints->n`, то в области графика на регулярной сетке точек `n x n` будут построены вектора поля.

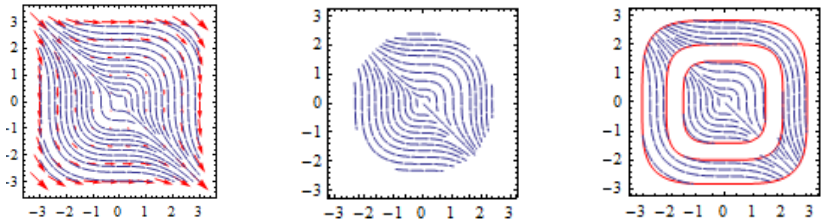
$$\mathbf{StreamPlot}[\{y^2,-x^2\},\{x,-3,3\},\{y,-3,3\},$$
$$\mathbf{VectorPoints}\to\mathbf{10},\mathbf{VectorStyle}\to\mathbf{Red}]$$

(следующий рисунок слева).

Область, в которой рисуются линии тока, можно ограничить, используя опцию `RegionFunction`

$$\mathbf{StreamPlot}[\{y^2,-x^2\},\{x,-3,3\},\{y,-3,3\},$$
$$\mathbf{RegionFunction}\to\mathbf{Function}[\{x,y,\mathrm{vx},\mathrm{vy},n\},x^2+y^2<6]]$$

(следующий рисунок в середине).



При этом ограничивающая область не обязана быть связной.

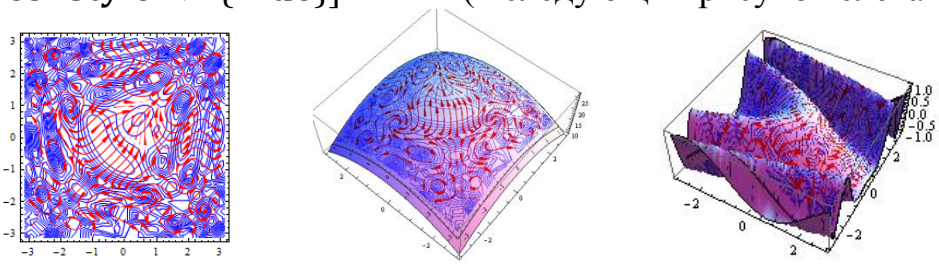
$$\mathbf{regionFn}=\mathbf{Function}[\{x,y,\mathrm{vx},\mathrm{vy},n\},4<n<8\ ||\ 0<n<2];$$
$$\mathbf{StreamPlot}[\{y^2,-x^2\},\{x,-3,3\},\{y,-3,3\},\mathbf{RegionFunction}\to\mathbf{regionFn},$$
$$\mathbf{BoundaryStyle}\to\mathbf{Red}]$$

(* предыдущий рисунок справа *)

Любопытна возможность использования графиков линий тока в качестве «поверхностных узоров». График линий тока, как и многие другие двумерные графические объекты, может быть наложен на поверхность как текстура.

$$\mathbf{strm}=\mathbf{StreamPlot}[\{\mathbf{Cos}[-1-x+y^2],\mathbf{Sin}[1+x^2-y]\},$$
$$\{x,-3,3\},\{y,-3,3\},\mathbf{StreamScale}\to\mathbf{0.25},$$
$$\mathbf{StreamStyle}\to\{\mathbf{Dart},\mathbf{Red}\},\mathbf{Mesh}\to\mathbf{10},$$
$$\mathbf{MeshStyle}->\{\mathbf{Blue}\}]$$

(* следующий рисунок слева *)



Используя директиву `Texture`, полученный «узор» накладывается на трехмерную поверхность

$$\mathbf{Plot3D}[27-x^2-y^2,\{x,-3,3\},\{y,-3,3\},\mathbf{Mesh}\to\mathbf{None},$$
$$\mathbf{PlotStyle}\to\mathbf{Texture}[\mathbf{strm}]]$$

(* предыдущий рисунок в середине *)

или на другую поверхность

$$\mathbf{Plot3D}[\mathbf{Sin}[27-x-y^2],\{x,-3,3\},\{y,-3,3\},\mathbf{Mesh}\to\mathbf{None},$$
$$\mathbf{PlotStyle}\to\mathbf{Texture}[\mathbf{strm}]]$$

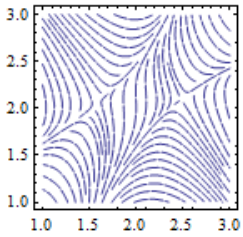
(* предыдущий рисунок справа *)

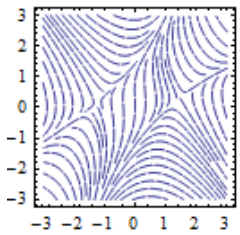
Опции, которые связаны с оформлением графиков и которые есть у функции `StreamPlot`, очень похожи на те, которые имеются у большинства

графических функций, в частности сходны с опциями функции `ContourPlot`. С большинством из них вы уже знакомы.

Если поле задано дискретно, т.е. задан набор векторов (пар чисел), то линии тока можно построить функцией `ListStreamPlot`. Она вначале выполняет интерполяцию переданных данных, а затем рисует линии тока, проходящие через регулярное множество точек в заданной области.

```
data = {{{-9, 7}, {0, -8}, {15, -20}}, {{9, 3}, {0, 2}, {6, 0}},
        {{20, -27}, {2, -9}, {-6, 9}}};
ListStreamPlot[data]    (* следующий рисунок слева *)
```





Обратите внимание на координаты, отложенные по осям графика. Они соответствуют индексам пар чисел (векторов). Если кроме векторов нужно указать точки их приложения, то список векторов надо дополнить координатами точек следующим образом.

```
data = {{{{-3, -3}, {-9, 7}}, {{-3, 0}, {0, -8}}, {{-3, 3}, {15, -20}}},
        {{{0, -3}, {9, 3}}, {{0, 0}, {0, 2}}, {{0, 3}, {6, 0}}},
        {{{3, -3}, {20, -27}}, {{3, 0}, {2, -9}}, {{3, 3}, {-6, 9}}}};
ListStreamPlot[data]          (* предыдущий рисунок справа *)
```

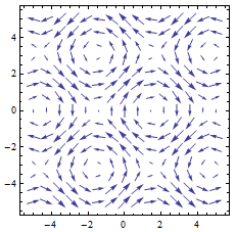
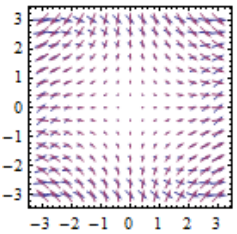
Например, первый вектор задан в виде {{-3,-3},{-9,7}} , где первая пара чисел представляет координаты точки, а вторая – координаты вектора. Общий вид поля не изменился, однако оно уже построено на реальных координатах точек.

Функция `ListStreamPlot` является аналогом функции `StreamPlot`. Во многом эти функции обладают идентичными возможностями и мы не будем здесь повторяться.

Другой способ визуализации двумерного векторного поля состоит в рисовании в плоской области стрелок, представляющих в некотором масштабе значение вектора $\vec{\mathbf{F}} = (P(x,y), Q(x,y))$ в этих точках. Такой рисунок в *Mathematica* создается функцией `VectorPlot`, которой в качестве аргументов передаются скалярные функции $P(x,y), Q(x,y)$.

В формате `VectorPlot[{P,Q},{x,`x_{min}`,`x_{max}`},{y`$_{min}$`,y`$_{max}$`}]` строится график векторного поля $(P(x,y), Q(x,y))$ на плоскости xy.

```
F[x_, y_] = {Sin[y], Cos[x]};
VectorPlot[F[x, y], {x, -5, 5}, {y, -5, 5}]    (* следующий  рисунок слева *)
```

Допустимо построение нескольких векторных полей на одном графике.

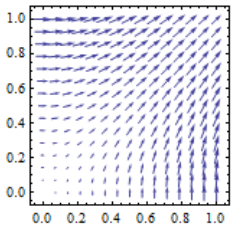
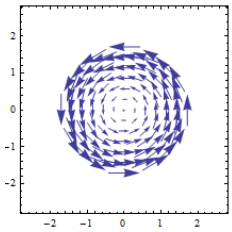
$$\mathbf{VectorPlot}[\{\{x^2+y^2, x^2-y^2\}, \{x, -y\}\}, \{x, -3, 3\}, \{y, -3, 3\}]$$

(предыдущий рисунок справа).

Используйте функцию `Evaluate` для символьного вычисления векторного поля перед его построением.

$$\mathbf{VectorPlot}[\mathbf{Evaluate}[\mathbf{Grad}[\mathbf{Sin}[xy], \{x, y\}]], \{x, 0, 1\}, \{y, 0, 1\}]$$

(следующий рисунок слева).

Попробуйте выполнить предыдущую команду без функции `Evaluate` – ничего не выйдет.

Область, в которой рисуется векторное поле, можно ограничить, используя опцию `RegionFunction`

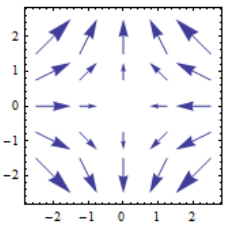
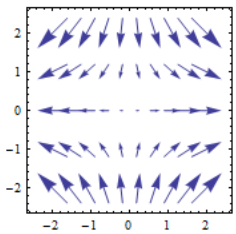
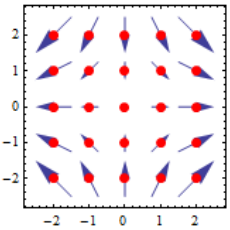
$$\mathbf{VectorPlot}[\{-y, x\}, \{x, -2, 2\}, \{y, -2, 2\}, \mathbf{VectorScale} \to .25,$$
$$\mathbf{RegionFunction} \to \mathbf{Function}[\{x, y, \mathrm{vx}, \mathrm{vy}, n\}, x^2+y^2 \le 3]]$$

(предыдущий рисунок справа).

Можно управлять количеством стрелок на графике с помощью опции `VectorPoints→n`. По умолчанию функция `VectorPlot` рисует вектора на регулярной сетке точек `n x n`.

$$\mathbf{VectorPlot}[\{-x, y\}, \{x, -2, 2\}, \{y, -2, 2\}, \mathbf{VectorPoints} \to 5]$$

(следующий рисунок слева)

Количество точек по разным осям можно задавать различным.

$$\mathbf{VectorPlot}[\{x, -y\}, \{x, -2, 2\}, \{y, -2, 2\}, \mathbf{VectorPoints} \to \{10, 5\}]$$

(предыдущий рисунок в середине).

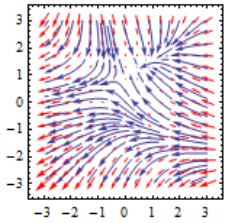
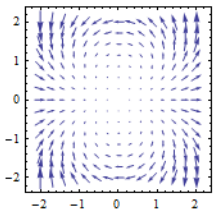
Точки, в которых будут рисоваться стрелочки, можно задавать самостоятельно.

$$\mathbf{points} = \mathbf{Flatten}[\mathbf{Table}[\{x, y\}, \{x, -2, 2, 1\}, \{y, -2, 2, 1\}], 1]$$
$$\mathbf{VectorPlot}[\{x, -y\}, \{x, -2, 2\}, \{y, -2, 2\}, \mathbf{VectorPoints} \to \mathbf{points},$$
$$\mathbf{VectorScale} \to .25, \mathbf{Epilog} \to \{\mathbf{Red}, \mathbf{PointSize}[\mathbf{Large}], \mathbf{Point}[\mathbf{points}]\}]$$

(предыдущий рисунок справа). Здесь опция `VectorScale` управляет размером стрелок.

Опция `StreamPoints->n` позволяет вместе с векторным полем нарисовать `n` линий тока.

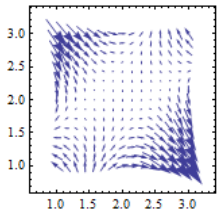
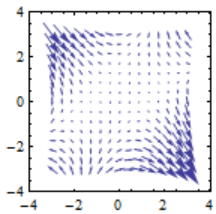
VectorPlot[{**−1 − x^2 + y, 1 + x − y^2**}, {**x, −3, 3**}, {**y, −3, 3**},
VectorStyle → Red, StreamPoints → 50,
VectorScale →. 1, StreamScale → 0. 2] (* следующий рисунок слева *)

Можно использовать комплексные функции
$z = x + yi$;
VectorPlot[{**Re**[z^2], **Im**[z^2]}, {**x, −2, 2**}, {**y, −2, 2**}, **VectorScale →. 1**]
(предыдущий рисунок справа).

Если поле задано дискретно, т.е. задан набор векторов (пар чисел), то векторное поле можно построить функцией `ListVectorPlot`. Вначале она выполняет интерполяцию переданных данных, а затем рисует стрелки векторов на регулярном множестве точек области.

data = {{{**−9, 7**}, {**0, −8**}, {**15, −20**}},
{{**9, 3**}, {**0, 2**}, {**6, 0**}},
{{**20, −27**}, {**2, −9**}, {**−6, 9**}}};
ListVectorPlot[**data, VectorScale → 0. 25**] (* следующий рисунок слева *)

Здесь опция `VectorScale` задает относительную длину стрелок.

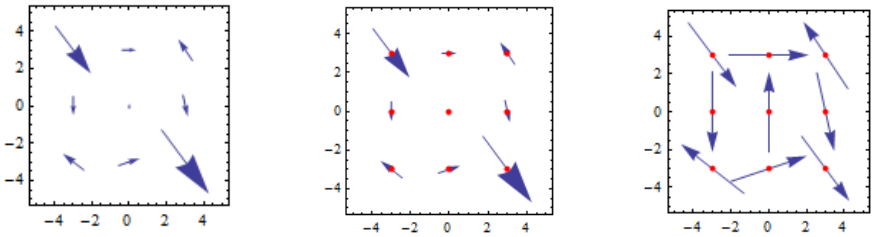
Если кроме векторов нужно задать точки их приложения, то список векторов надо дополнить координатами точек следующим образом.

data = {{{{**−3, −3**}, {**−9, 7**}}, {{**−3, 0**}, {**0, −8**}}, {{**−3, 3**}, {**15, −20**}}},
{{{**0, −3**}, {**9, 3**}}, {{**0, 0**}, {**0, 2**}}, {{**0, 3**}, {**6, 0**}}},
{{{**3, −3**}, {**20, −27**}}, {{**3, 0**}, {**2, −9**}}, {{**3, 3**}, {**−6, 9**}}}};
ListVectorPlot[**data, VectorScale → 0. 2**] (* предыдущий рисунок справа *)

Графики на предыдущем рисунке отличаются только значениями, указанными на координатных осях.

Можно изобразить только те вектора, которые были преданы. Для этого используется значение опции `VectorPoints→All`.

ListVectorPlot[**data, VectorScale → 0. 5, VectorPoints → All**]
(следующий рисунок слева).



Можно изобразить вектора вместе с точками их приложения

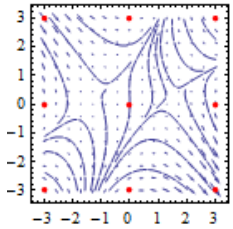
```
points = Flatten[Table[data[[j, i, 1]], {j, 3}, {i, 3}], 1];
ListVectorPlot[data, VectorScale → 0.5, VectorPoints → All,
  Epilog → {Red, PointSize[0.03], Point[points]}] (* пред. рис. в середине *)
```

Можно рисовать стрелочки постоянного размера (те же значения **data** и **points**, что и в предыдущем коде).

```
ListVectorPlot[data, VectorScale → {0.5, 0.5, None}, VectorPoints → All,
  Epilog → {Red, PointSize[0.03], Point[points]}]   (* пред. рис. справа *)
```

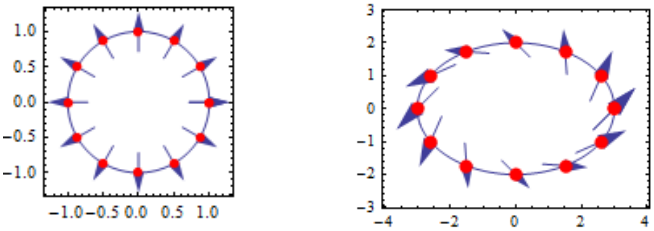
Вместе с векторным полем можно рисовать линии тока поля. Для этого используется опция `StreamPoints`.

```
points = Flatten[Table[data[[j, i, 1]], {j, 3}, {i, 3}], 1];
ListVectorPlot[data, Epilog → {Red, PointSize[0.03], Point[points]},
  StreamPoints → 20]
```



Пример. Нарисовать поле нормалей единичной окружности.

```
p = {x, y};
v = {x, y};                          (* векторное поле *)
r[t_] = {Cos[t], Sin[t]};            (* параметрическое уравнение кривой *)
dt = {p, v} /. Thread[p → r[t]];
data = Table[dt, {t, 0, 2π, π/6}];   (* точки вместе с векторами *)
points = Table[data[[i, 1]], {i, Length[data]}];   (* только точки *)
lvp = ListVectorPlot[data, VectorPoints → All, VectorScale → 0.1,
  Epilog → {Red, PointSize[0.02], Point[points]}];
ln = ParametricPlot[r[t], {t, 0, 2π}];
Show[lvp, ln]                        (* следующий рисунок слева *)
```



Если вы измените вторую и третью строки предыдущего кода, например так

```
v = {x - y, x + y};
r[t_] = {3Cos[t], 2Sin[t]};
```

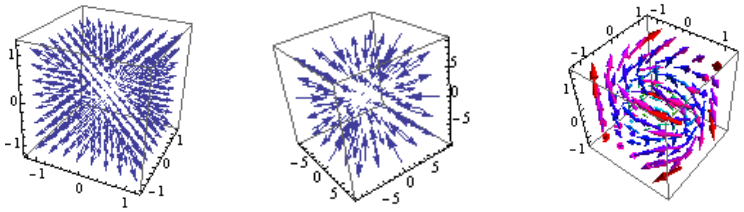
то постройте другое векторное поле на другой кривой (предыдущий рисунок справа).

□

Многие опции функций `VectorPlot` и `ListVectorPlot` имеют такой же смысл как и у функции `Graphics` и мы не будем здесь повторяться, поясняя их значение.

Для построения векторных полей в трехмерном пространстве используется функция `VectorPlot3D`.

VectorPlot3D[{x, y, z}, {$x, -1, 1$}, {$y, -1, 1$}, {$z, -1, 1$}] (* след. рисунок слева *)



Используйте функцию `Evaluate` для символьного вычисления векторного поля перед его построением.

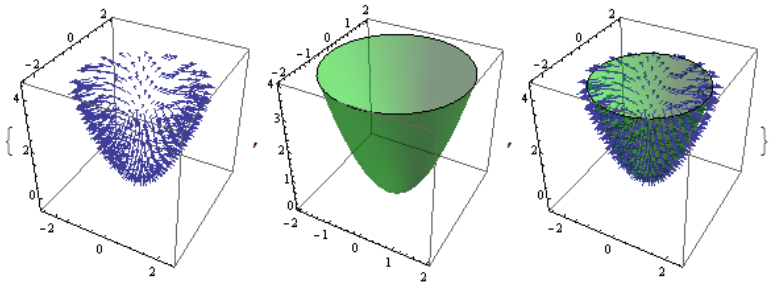
VectorPlot3D[**Evaluate**[D[**Sin**[$x^2 + y^2 + z^2$], {{x, y, z}}]],
{$x, -2\pi, 2\pi$}, {$y, -2\pi, 2\pi$}, {$z, -2\pi, 2\pi$}, **VectorPoi**nts → 5]
(предыдущий рисунок в середине).

Можно управлять цветом и формой стрелок. В следующей команде форма стрелок задается опцией `VectorStyle→"Arrow3D"`, а опция `VectorColorFunction→Hue` задает цвет в соответствии с нормой (`Norm`) векторного поля.

VectorPlot3D[{$y, -x, z$}, {$x, -1, 1$}, {$y, -1, 1$}, {$z, -1, 1$},
PlotRange → **All**, **VectorPoints** → 5, **VectorColorFunction** → **Hue**,
VectorStyle → "**Arrow3D**"] (* предыдущий рисунок справа *).

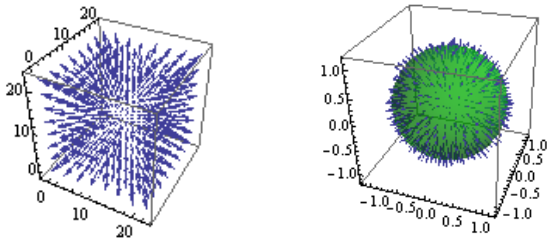
В следующем примере мы рисуем поле градиента скалярного поля $u = x^2 + y^2 - z$ в окрестности поверхности $u = 0$. Для этого мы используем опцию `RegionFunction`. С ее помощью мы выделяем узкую область векторного поля в окрестности этой поверхности.

scalarField = $x^2 + y^2 - z$;
vectorField = D[**scalarField**, {{x, y, z}}];
v = **VectorPlot3D**[**vectorField**, {$x, -2, 2$}, {$y, -2, 2$}, {$z, 0, 4$},
VectorPoints → 25, **VectorScale** → {0.15, **Scaled**[0.5]},
RegionFunction → **Function**[{x, y, z}, $-0.1 \le$ **scalarField** <= 0.1]];
c = **ContourPlot3D**[**scalarField** == 0, {$x, -2, 2$}, {$y, -2, 2$}, {$z, 0, 4$},
Mesh → **None**, **ContourStyle** → **Opacity**[0.5, **Green**]];
{v, c, **Show**[v, c]}



Если векторное поле задано дискретно, т.е. задан набор векторов (троек чисел), то векторное поле можно построить функцией `ListVectorPlot3D`. Как и другие аналогичные функции она вначале выполняет интерполяцию данных, а потом рисует стрелки векторов.

ListVectorPlot3D[Table[{*x*, *y*, *z*}, {*x*, −1, 1, . 1}, {*y*, −1, 1, . 1}, {*z*, −1, 1, . 1}]]
(следующий рисунок слева).



Обратите внимание на координаты точек векторного поля. Они представляются индексами векторов {*x*,*y*,*z*}, а не значениями *x*,*y*,*z*.

В следующем коде мы изображаем векторное поле нормалей к сфере. Но с помощью опции `DataRange` это поле отнесено к реальным координатам точек, а не к индексам векторов.

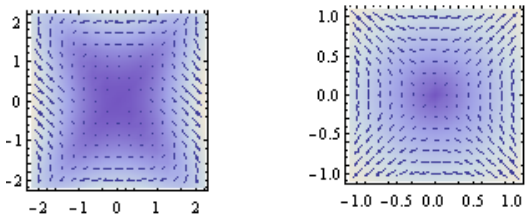
```
scalarField = x^2 + y^2 + z^2 - 1;
vectorField = D[scalarField, {{x, y, z}}];
data = Table[vectorField, {x, -1, 1}, {y, -1, 1}, {z, -1, 1}];
v = ListVectorPlot3D[data, DataRange → {{-1, 1}, {-1, 1}, {-1, 1}},
      VectorPoints → 15, VectorScale → {0. 1, Scaled[0. 5]},
      RegionFunction → Function[{x, y, z}, -0. 1 ≤ scalarField <= 0.1]];
c = ContourPlot3D[scalarField == 0, {x, -1, 1}, {y, -1, 1}, {z, -1, 1},
      Mesh → None, ContourStyle → Opacity[0. 5, Green]];
Show[v, c]   (* предыдущий рисунок справа *)
```

Для одновременной визуализации скалярного и векторного поля можно использовать функции `VectorDensityPlot` и `StreamDensityPlot`. Они строят график векторного поля или его линий тока на фоне графика плотности нормы этого поля (или другой скалярной функции, которую укажет пользователь). Фон раскрашивается в цвета, яркость которых соответствует значению этого скалярного поля. В формате

`VectorDensityPlot[{vx, vy}, {x, xmin, xmax}, {ymin, ymax}]`

рисуются стрелочки векторного поля, а скалярной функцией, значения которой используются для графика плотности, является норма векторного поля `{vx, vy}`.

VectorDensityPlot[{*x*^2 − *y*^2, −*x*^2}, {*x*, −2, 2}, {*y*, −2, 2}] (* след. рис. слева *)



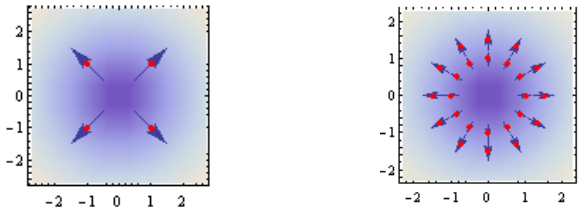
Используйте функцию `Evaluate` для символьного вычисления векторного поля перед его построением

$$\mathbf{VectorDensityPlot[Evaluate[}\boldsymbol{D[xy,\{\{x,y\}\}]]\mathbf{,}\{x,-1,1\}\mathbf{,}\{y,-1,1\}\mathbf{]}$$

(предыдущий рисунок справа).

Можно явно указать точки, в которых должны быть нарисованы стрелочки. Для этого используется опция `VectorPointe→точки`.

```
points = {{-1, -1}, {-1, 1}, {1, -1}, {1, 1}};
VectorDensityPlot[{x, y}, {x, -2, 2}, {y, -2, 2},
    VectorPoints → points, VectorScale →.25,
    Epilog → {Red, PointSize[Medium], Point[points]}] (* след. рис. слева *)
```



В качестве примера изобразим вектора поля только на двух окружностях

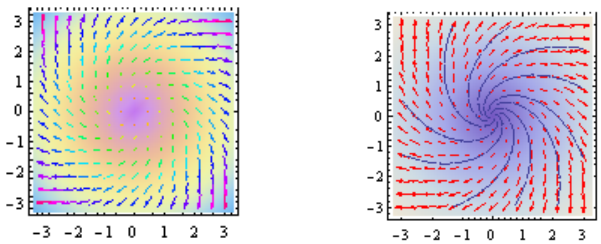
```
points = Join[Table[{Cos[t], Sin[t]}, {t, 0, 2π, π/6}],
              Table[{1.5Cos[t], 1.5Sin[t]}, {t, 0, 2π, π/6}]];
VectorDensityPlot[{x, y}, {x, -2, 2}, {y, -2, 2},
              VectorPoints → points, VectorScale → Large,
              Epilog → {Red, PointSize[Medium], Point[points]}]
```

(предыдущий рисунок справа).

Опции `ColorFunction` и `VectorColorFunction` позволяют управлять цветом фона и стрелочек. В следующем примере значением, которое используется для выбора цвета является норма векторного поля.

```
VectorDensityPlot[{y + x, y - x}, {x, -3, 3}, {y, -3, 3},
        ColorFunction → "Pastel", VectorColorFunction → Hue]
```

(следующий рисунок слева).



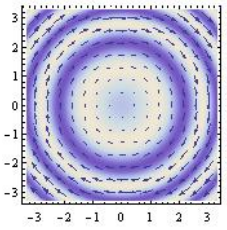
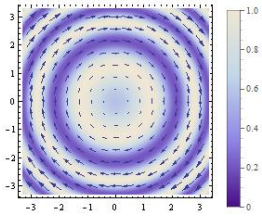
Можно рисовать векторное поле и линии его тока совместно. Для этого используется опция `StreamPoints→n`, где `n` количество линий тока.

```
VectorDensityPlot[{x + y, y - x}, {x, -3, 3}, {y, -3, 3},
        VectorStyle → Red, StreamPoints → 10]  (* пред. рис. справа *)
```

В формате

VectorDensityPlot[{{v_x, v_y}, s}, {x, x_{min}, x_{max}}, {y_{min}, y_{max}}]
выражение s определяет скалярную функцию, значения которой будут использоваться при построении графика плотности (фона).

$$\mathbf{VectorDensityPlot}[\{\{y,-x\},\mathbf{Sin}[x^2+y^2]\},\{x,-3,3\},\{y,-3,3\},\ \mathbf{MaxRecursion}\to 3]$$ (* следующий рисунок слева *)

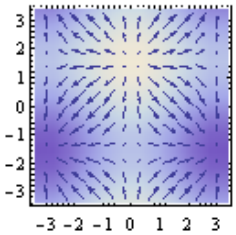
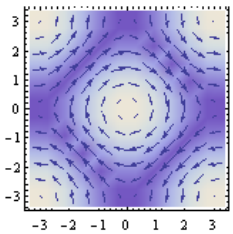
У функции VectorDensityPlot имеется возможность построения шкалы цветов.

$$\mathbf{VectorDensityPlot}[\{\{y,-x\},\mathbf{Sin}[x^2+y^2]\},\{x,-3,3\},\{y,-3,3\},\ \mathbf{MaxRecursion}\to 3,\mathbf{PlotLegends}->\mathbf{BarLegend}[\{\text{"LakeColors"},\{0,1\}\}]]$$
(предыдущий рисунок справа).

Построим векторное поле на фоне его дивергенции

$$\boldsymbol{f}=\{\mathbf{Sin}[x],-\mathbf{Cos}[y]\};$$
$$\boldsymbol{s}=\mathbf{Div}[\boldsymbol{f},\{x,y\}]$$
$$\mathbf{VectorDensityPlot}[\{\boldsymbol{f},\boldsymbol{s}\},\{x,-3,3\},\{y,-3,3\},\ \mathbf{VectorPoints}\to 12]$$ (* следующий рисунок слева *)

$\mathrm{Cos}[x]+\mathrm{Sin}[y]$

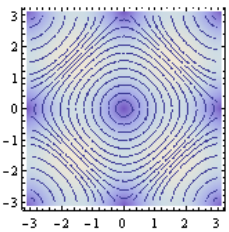
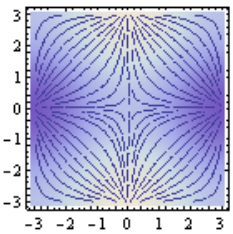
Построим векторное поле на фоне нормы его ротора

$$\boldsymbol{f}=\{\mathbf{Sin}[y],-\mathbf{Sin}[x]\};$$
$$\mathbf{crl}=\mathbf{Curl}[\boldsymbol{f},\{x,y\}]$$
$$\mathbf{VectorDensityPlot}[\{\boldsymbol{f},\mathbf{Norm}[\mathbf{crl}]\},\{x,-3,3\},\{y,-3,3\},\ \mathbf{VectorPoints}\to 12]$$ (* предыдущий рисунок справа *)

$-\mathrm{Cos}[x]-\mathrm{Cos}[y]$

Функция StreamDensityPlot строит линии тока на фоне графика плотности некоторого скалярного поля. По умолчанию этим скалярным полем является норма векторного поля.

$$\mathbf{StreamDensityPlot}[\{\mathbf{Sin}[y],-\mathbf{Sin}[x]\},\{x,-3,3\},\{y,-3,3\}]$$
(следующий рисунок слева)

В формате

StreamDensityPlot[{{v_x, v_y}, s}, {x, x_{min}, x_{max}}, {y_{min}, y_{max}}]

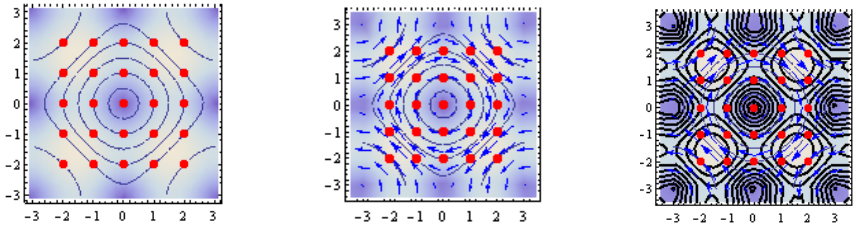
выражение s задает скалярную функцию, значения которой будут использоваться при построении графика плотности. В качестве примера построим линии тока поля на фоне его дивергенции.

```
f = {Sin[x], -Sin[y]};
s = Div[f, {x, y}]
StreamDensityPlot[{f, s}, {x, -3, 3}, {y, -3, 3}]  (* предыдущий рис. справа *)
Cos[x] - Cos[y]
```

Построим линии тока, которые проходят через заданные точки. Для этого используем опцию StreamPoints→точки.

```
points = Tuples[{-2, -1, 0, 1, 2}, 2];
StreamDensityPlot[{Sin[y], -Sin[x]}, {x, -3, 3}, {y, -3, 3},
    StreamPoints → points, Epilog → {Red, PointSize[0.02], Point[points]}]
```

(следующий рисунок слева)



К графику линий тока можно добавить график векторного поля (стрелочки).

```
StreamDensityPlot[{Sin[y], -Sin[x]}, {x, -3, 3}, {y, -3, 3},
    StreamPoints → points,
    Epilog → {Red, PointSize[0.04], Point[points]},
    VectorPoints → 10, VectorStyle → Blue]  (* пред. рис. в середине *)
```

Имеется также опция Mesh, которая позволяет рисовать линии уровня некоторой скалярной функции. По умолчанию такой функцией является норма векторного поля.

```
StreamDensityPlot[{Sin[y], -Sin[x]}, {x, -3, 3}, {y, -3, 3},
    StreamPoints → points, VectorPoints → 10, VectorStyle → Blue,
    Epilog → {Red, PointSize[0.04], Point[points]},
    Mesh → 10, MeshStyle → Thick]  (* предыдущий рисунок справа *)
```

Для дискретных наборов данных аналогами функций VectorDensityPlot и StreamDensityPlot являются функции ListVectorDensityPlot и ListStreamDensityPlot. Возможности этих функций сходны и мы не будем на них останавливаться.

3.5 Ряды

3.5.1 Вычисление конечных и бесконечных сумм.

Для символьного вычисления конечных и бесконечных сумм используется функция Sum. В формате Sum[f,{i,i_{max}}] она вычисляет сумму вида $\sum_{i=1}^{i_{max}} f$.

$$\mathbf{Sum}[\mathbf{2k-1},\{\mathbf{k},\mathbf{12}\}]$$

144

Sum[f,{i,i_{max}}] может быть введена как $\sum_{i}^{i_{max}} f$. Трафарет ввода суммы можно найти на панели специальных символов «Basic Math Input». Можно также использовать специальные комбинации клавиш. Для ввода знака суммы» наберите Esc−sum−Esc. Потом надо ввести подстрочный и надстрочный индексы у значка суммы Σ. Для этого введите Ctrl−_ (подчеркивание), затем Ctrl−% (или Ctrl−^, затем Ctrl−%). Введите значения нижнего и верхнего пределов суммирования. Ctrl−пробел возвращает курсор на нормальный уровень ввода, где введите суммируемое выражение. Следующая команда эквивалентна предыдущей.

$$\sum_{\mathbf{k}}^{\mathbf{12}} (\mathbf{2k-1})$$

144

В формате Sum[f,{i,i_{min},i_{max}}] задается начальное $i = i_{min}$ и конечное i_{max} значение счетчика.

$$\mathbf{Sum}[\mathbf{k},\{\mathbf{k},\mathbf{4},\mathbf{8}\}]$$

30

или

$$\sum_{\mathbf{k=4}}^{\mathbf{8}} \mathbf{k}$$

30

В формате Sum[f,{i,i_{min},i_{max},di}] задается еще шаг счетчика di.

$$\mathbf{Sum}[\mathbf{k},\{\mathbf{k},\mathbf{4},\mathbf{12},\mathbf{3}\}]$$

21

В формате Sum[f,{i,i_{min},i_{max}},{j_{min},j_{max}}] вычисляется двукратная сумма вида $\sum_{i=i_{min}}^{i_{max}} \sum_{j=j_{min}}^{j_{max}} f$.

$$\mathbf{Sum}[(\boldsymbol{i}+\boldsymbol{j})^{\mathbf{2}},\{\boldsymbol{i},\mathbf{3}\},\{\boldsymbol{j},\mathbf{4}\}]$$

266

Допустимо суммирование по большему количеству переменных/индексов.

Можно суммировать выражения, содержащие более чем одну переменную; другие переменные будут трактоваться, как константы.

$$\sum_{\mathbf{k=1}}^{\mathbf{7}} \frac{\boldsymbol{x}^{\boldsymbol{k}}}{\boldsymbol{k}!}$$

$$x+\frac{x^2}{2}+\frac{x^3}{6}+\frac{x^4}{24}+\frac{x^5}{120}+\frac{x^6}{720}+\frac{x^7}{5040}$$

Как и выше можно задать шаг изменения переменной суммирования.

$$\mathbf{Sum}\left[\frac{\boldsymbol{x^k}}{\boldsymbol{k!}},\{\boldsymbol{k},\mathbf{1},\mathbf{7},\mathbf{2}\}\right]$$

$$x+\frac{x^3}{6}+\frac{x^5}{120}+\frac{x^7}{5040}$$

В двойной сумме «внутренние» пределы суммирования могут зависеть от внешнего индекса.

$$\sum_{\mathbf{m}=\mathbf{1}}^{\mathbf{3}}\sum_{\boldsymbol{n}=\mathbf{1}}^{\mathbf{m}}\mathbf{Sin}[\boldsymbol{x^m}+\boldsymbol{y^n}]$$

$$\mathrm{Sin}[x+y]+\mathrm{Sin}[x^2+y]+\mathrm{Sin}[x^3+y]+\mathrm{Sin}[x^2+y^2]+\mathrm{Sin}[x^3+y^2]+\mathrm{Sin}[x^3+y^3]$$

Результат вычисления суммы может быть представлен символически.

$$\sum_{\boldsymbol{k}=\mathbf{1}}^{\boldsymbol{n}}\boldsymbol{k^4}$$

$$\frac{1}{30}n(1+n)(1+2n)(-1+3n+3n^2)$$

$$\sum_{\mathbf{k}=\mathbf{1}}^{\mathbf{n}}\frac{(-\mathbf{1})^{\boldsymbol{k}}\boldsymbol{k}}{\boldsymbol{4k^2-1}}$$

$$\frac{-1+(-1)^n-2n}{4(1+2n)}$$

$$\mathbf{Sum}[\frac{\mathbf{1}}{\boldsymbol{i}(\boldsymbol{i}+\mathbf{1})(\boldsymbol{i}+\mathbf{2})},\{\boldsymbol{i},\boldsymbol{n}\}]$$

$$\frac{1}{4}-\frac{1}{2(1+n)(2+n)}$$

Пределы суммирования могут быть бесконечными (`Infinity`). Такие суммы принято называть рядами и *Mathematica* умеет их вычислять.

$$\sum_{\mathbf{k}=\mathbf{1}}^{\infty}\frac{\mathbf{1}}{\boldsymbol{k^2}}$$

$$\frac{\pi^2}{6}$$

$$\sum_{\mathbf{k}=\mathbf{1}}^{\infty}\frac{(-\mathbf{1})^{\boldsymbol{k}+\mathbf{1}}}{(\mathbf{2}\boldsymbol{k}-\mathbf{1})^{\mathbf{5}}}$$

$$\frac{5\pi^5}{1536}$$

$$\sum_{\mathbf{k}=\mathbf{1}}^{\infty}\frac{(-\mathbf{1})^{\boldsymbol{k}-\mathbf{1}}\boldsymbol{k}}{(\boldsymbol{k}+\mathbf{1})^{\mathbf{2}}}$$

$$\frac{\pi^2}{12}-\mathrm{Log}[2]$$

$$\sum_{k=0}^{\infty} \frac{1}{k!\,(k+z)!}$$

BesselI[z, 2]

Множество, которое пробегает индекс суммирования, можно задавать конечным списком.

Sum[$f[i]$, {i, {a, b, c}}]

$f[a] + f[b] + f[c]$

Если найти результат суммирования в символьной форме не удается, *Mathematica* возвращает сумму невычисленной.

Sum[$n/(n! + 1)$, {n, Infinity}]

$$\sum_{n}^{\infty} \frac{n}{1+n!}$$

Однако, для числовых рядов можно получить приближенное значение суммы, используя функцию `N`.

N[%]

```
1.8064971
```

Применение функции `N` к результату, полученному с помощью функции `Sum`, эквивалентно использованию функции `NSum`, которая служит для численного суммирования рядов. Для бесконечных числовых рядов результат получается только в случае, если ряд сходится.

NSum[n/(n! + 1), {n, Infinity}]

```
1.8064971
```

ns = NSum[(−3)^i/i!, {i, 0, ∞}]

```
0.049787068
```

Последняя сумма вычисляется символьно

ts = Sum[(−3)^i/i!, {i, 0, ∞}]

$$\frac{1}{e^3}$$

и ошибка вычислений составляет

ns − ts

```
3.469447×10^-17
```

У функции `NSum` имеются опции, которые управляют точностью вычислений. `AccuracyGoal` определяет абсолютную погрешность вычислений, используя количество значащих цифр. `PrecisionGoal->n` определяет для значения x погрешность вычислений равную $|x|10^{-n}$. Когда включены опции `AccuracyGoal->m` и `PrecisionGoal->n`, то *Mathematica* пытается выполнять вычисления величины x с погрешностью $\max\left(10^{-m}, |x|10^{-n}\right)$. Опция `WorkingPrecision` определяет количество значащих цифр, используемых во внутренних вычислениях; установка `WorkingPrecision→ MachinePrecision` приводит к вычислениям с процессорной точностью.

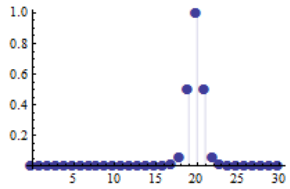
Использование этих опций идентично их использованию для функции NIntegrate.

$$\mathbf{ns = NSum[(-3)\text{^}i/i!, \{i, 0, \infty\}, WorkingPrecision \to 4]}$$

```
0.04979
```

По умолчанию функция NSum суммирует первые 15 членов перед тем, как переходит к аппроксимации «хвоста» ряда. Однако это не всегда корректно. Например, построим график следующей последовательности

$$\mathbf{DiscretePlot}\left[\frac{1}{(k-20)^4+1}, \{k, 0, 30\}, \mathbf{PlotRange \to All}, \mathbf{PlotMarkers \to \{Automatic, Medium\}}\right]$$



Максимальное значение последовательность достигает на 20 – м члене. Поэтому суммирование 15 – ти членов с последующей аппроксимацией «хвоста» будет иметь большую погрешность

$$\mathbf{NSum}\left[\frac{1}{(k-20)^4+1}, \{k, 0, \infty\}\right]$$

```
2.2214029
```

Чтобы вначале просуммировать большее количество элементов используется опция NSumTerms.

$$\mathbf{NSum}\left[\frac{1}{(k-20)^4+1}, \{k, 0, \infty\}, \mathbf{NSumTerms \to 30}\right]$$

```
2.1569165
```

По умолчанию функция NSum выполняет проверку сходимости ряда. Если вы заранее знаете, что ряд сходится, то отключив проверку сходимости опцией VerifyConvergence→False можно ускорить вычисления.

Если ряд не сходится, то функция NSum в качестве результата возвращает ComplexInfinity.

$$\mathbf{NSum[1, \{k, 1, \infty\}]}$$

```
ComplexInfinity
```

Но часто она выводит «угрожающие» сообщения, и представляет какой – то численный результат, которому нельзя доверять.

$$\mathbf{NSum}\left[1/\sqrt{k}, \{k, 1, \infty\}\right]$$

```
Numerical integration converging too slowly...
3.8439417962027545^10^13977
```

В *Mathematica* имеются функции Product и NProduct, вычисляющие произведения $\prod_{i=i_{\min}}^{i_{\max}} f$. Обращение к ним аналогично формату обращения к функциям Sum и NSum.

$$\mathbf{Product}[i, \{i, n\}]$$

$$n!$$

$\boldsymbol{p = 4}$;

$$\mathbf{HoldForm}\left[\mathbf{Sum}\left[\frac{\mathbf{1}}{\prod_{\boldsymbol{i=0}}^{\boldsymbol{p}}(\boldsymbol{\alpha+n+i})}, \{\boldsymbol{n}, \mathbf{1}, \infty\}\right]\right] ==$$

$$\mathbf{Sum}[\frac{\mathbf{1}}{\prod_{\boldsymbol{i=0}}^{\boldsymbol{p}}(\boldsymbol{\alpha+n+i})}, \{\boldsymbol{n}, \mathbf{1}, \infty\}]$$

$$\sum_{n=1}^{\infty} \frac{1}{\prod_{i=0}^{p}(\alpha+n+i)} == \frac{1}{4(1+\alpha)(2+\alpha)(3+\alpha)(4+\alpha)}$$

Здесь функция `HoldForm` используется для того, чтобы в левой части записать выражение в невычисленном виде.

Часто результат символьного суммирования зависит от значений параметров, стоящих в суммируемом выражении. Если при каких – то значениях параметров ряд сходится, то возвращается результат, соответствующий таким значениям. Если вы хотите увидеть условия, при которых этот результат верен, то следует использовать опцию `GenerateConditions→True`.

$$\mathbf{Sum}[\boldsymbol{a^n}, \{\boldsymbol{n}, \mathbf{0}, \infty\}, \mathbf{GenerateConditions} \rightarrow \mathbf{True}]$$

$$\text{ConditionalExpression}[\frac{1}{1-a}, \text{Abs}[a] < 1]$$

Если в полученном результате вы желаете избавиться от предположений, то используйте функцию `Refine`.

$$\mathbf{Refine}[\%, \mathbf{Abs}[\boldsymbol{a}] < 1]$$

$$\frac{1}{1-a}$$

Если опцию `GenerateConditions→True` не использовать, то сразу получим результат (без предположений, хотя они использовались при вычислении).

$$\mathbf{Sum}[\boldsymbol{a^n}, \{\boldsymbol{n}, \mathbf{0}, \infty\}]$$

$$\frac{1}{1-a}$$

Разложение не всегда получается таким, как хочется

$$\mathbf{sm} = \mathbf{Sum}[\frac{\mathbf{Sin}[\boldsymbol{n\,x}]}{\boldsymbol{n}}, \{\boldsymbol{n}, \mathbf{1}, \infty\}]$$

$$\frac{1}{2}\,i\,(\text{Log}[1 - e^{ix}] - \text{Log}[e^{-ix}(-1 + e^{ix})])$$

Для получения более простого выражения следует выполнить алгебраические преобразования результата

$$\mathbf{logrule} = \mathbf{Log}[\mathbf{a\_}] - \mathbf{Log}[\mathbf{b\_}] \rightarrow \mathbf{Log}[\boldsymbol{a/b}];$$

$$\mathbf{sm1} = \mathbf{Simplify}[\mathbf{sm}/.\,\mathbf{logrule}]$$

$$\mathbf{PowerExpand}[\mathbf{sm1}, \mathbf{Assumptions} :\rightarrow \mathbf{0} < \boldsymbol{x} < 2\boldsymbol{\pi}]$$

$$\frac{\pi}{2} - \frac{x}{2}$$

В выражении **sm** мы выполнили подстановку Log[*a*_]–Log[*b*_]→Log[*a*/*b*], а следом использовали функцию `PowerExpand` для выполнения

преобразования `Log[a^b]` в `bLog[a]`, которое автоматически функцией `Simplify` не выполняется. Т.о. имеет место равенство $\sum_{n=1}^{\infty}\frac{\sin n\,x}{n}=\frac{\pi-x}{2}$.

3.5.2 Ряды Тейлора.

Произвольную функцию `f(x)` можно разложить в степенной ряд около точки $x=x_0$, используя функцию `Series`. В формате `Series[f,{x,x0,n}]` она генерирует степенной ряд $\sum_{k=0}^{n}\frac{f^{(k)}(x_0)}{k!}(x-x_0)^k$ в окрестности точки x_0 до слагаемого $(x-x_0)^n$. Например,

$\mathbf{Series[}\boldsymbol{f}\mathbf{[}\boldsymbol{x}\mathbf{],\{}\boldsymbol{x}\mathbf{,0,5\}]}$

$f[0]+f'[0]x+\frac{1}{2}f''[0]x^2+\frac{1}{6}f^{(3)}[0]x^3+\frac{1}{24}f^{(4)}[0]x^4+\frac{1}{120}f^{(5)}[0]x^5+O[x]^6$

$\mathbf{g=Series[Exp[}\boldsymbol{x}\mathbf{],\{}\boldsymbol{x}\mathbf{,0,5\}]}$

$1+x+\frac{x^2}{2}+\frac{x^3}{6}+\frac{x^4}{24}+\frac{x^5}{120}+O[x]^6$

$\mathbf{Series[}\frac{\mathbf{Sin[}\boldsymbol{x}\mathbf{]}}{\boldsymbol{x}}\mathbf{,\{}\boldsymbol{x}\mathbf{,0,8\}]}$

$1-\frac{x^2}{6}+\frac{x^4}{120}-\frac{x^6}{5040}+\frac{x^8}{362880}+O[x]^9$

Как видите, в конце полинома, представляющего ряд, функция `Series` выводит символ $O\left[x^{n+1}\right]$. Следует помнить, что функция `Series` создает ненулевой остаточный член, даже если он тождественно равен нулю.

$\mathbf{Series[}\boldsymbol{x}^{\mathbf{5}}\mathbf{,\{}\boldsymbol{x}\mathbf{,0,5\}]}$

$x^5+O[x]^6$

$\boldsymbol{P}\mathbf{[x\_]}=\boldsymbol{x}^{\mathbf{5}}+\mathbf{2}\boldsymbol{x}^{\mathbf{4}}-\mathbf{3}\boldsymbol{x}^{\mathbf{3}}+\mathbf{6}\boldsymbol{x}^{\mathbf{2}}-\mathbf{12}\boldsymbol{x}+\mathbf{17};$

$\mathbf{Series[}\boldsymbol{P}\mathbf{[}\boldsymbol{x}\mathbf{],\{}\boldsymbol{x}\mathbf{,0,5\}]}$

$17-12x+6x^2-3x^3+2x^4+x^5+O[x]^6$

Функцию `Series` можно использовать для переразложеня полинома по степеням $x-a$ при любом a.

$\mathbf{Series[}\boldsymbol{P}\mathbf{[}\boldsymbol{x}\mathbf{],\{}\boldsymbol{x}\mathbf{,1,5\}]}$

$11+4(x-1)+19(x-1)^2+15(x-1)^3+7(x-1)^4+(x-1)^5+O[x-1]^6$

С рядами можно выполнять обычные арифметические операции.

$\mathbf{G=g\text{\^{}}2(1+g)}$ (* **g** это разложение в ряд экспоненты, см. выше *)

$2+5x+\frac{13x^2}{2}+\frac{35x^3}{6}+\frac{97x^4}{24}+\frac{55x^5}{24}+O[x]^6$

$\frac{\mathbf{Series[Log[1+}\boldsymbol{x}\mathbf{],\{}\boldsymbol{x}\mathbf{,0,7\}]}}{\mathbf{Series[Log[1+}\boldsymbol{x}^{\mathbf{2}}\mathbf{],\{}\boldsymbol{x}\mathbf{,0,7\}]}}$

$\frac{1}{x}-\frac{1}{2}+\frac{5x}{6}-\frac{x^2}{2}+\frac{17x^3}{60}-\frac{x^4}{4}+O[x]^5$

$\boldsymbol{t}=\mathbf{Series[ArcTan[}\boldsymbol{x}\mathbf{],\{}\boldsymbol{x}\mathbf{,0,8\}]}$

$$x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + O[x]^9$$

t^2

$$x^2 - \frac{2x^4}{3} + \frac{23x^6}{45} - \frac{44x^8}{105} + O[x]^{10}$$

Можно разлагать в степенные ряды функции нескольких переменных

Series[Sin[xy], {x, 0, 5}, {y, 0, 5}]

$$(y + O[y]^6)x + (-\frac{y^3}{6} + O[y]^6)x^3 + (\frac{y^5}{120} + O[y]^6)x^5 + O[x]^6$$

Функция `Series` создает объект `SeriesData`, который можно использовать как функцию.

scos = Series[Cos[x], {x, 0, 5}]

$$1 - \frac{x^2}{2} + \frac{x^4}{24} + O[x]^6$$

scos * x^2

$$x^2 - \frac{x^4}{2} + \frac{x^6}{24} + O[x]^8$$

scos + (x^2 - x - 1)

$$-x + \frac{x^2}{2} + \frac{x^4}{24} + O[x]^6$$

D[scos, x]

$$-x + \frac{x^3}{6} + O[x]^5$$

Однако вычислить значение выражения или построить график не поучается.

scos/. x → 1

```
SeriesData::ssdn: Attempt to evaluate a series at the number 1.
Returning Indeterminate.
Indeterminate
```

Чтобы это сделать надо отбросить остаточный член. Функция `Normal` оставляет только степенной многочлен, отбрасывая символ $O\left[x^{n+1}\right]$.

log1 = Series[Log[1 + x], {x, 0, 7}]

$$x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \frac{x^5}{5} - \frac{x^6}{6} + \frac{x^7}{7} + O[x]^8$$

Normal[log1]

$$x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \frac{x^5}{5} - \frac{x^6}{6} + \frac{x^7}{7}$$

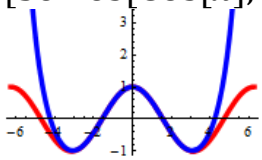
С полученными многочленами можно уже выполнять стандартные операции без всяких ограничений.

Factor[%]

$$\frac{1}{420}x(420 - 210x + 140x^2 - 105x^3 + 84x^4 - 70x^5 + 60x^6)$$

После отбрасывания «остаточного члена» можно строить график полинома, приближающего исходную функцию. Вот пример графика функции Cos[x] и ее ряда Тейлора до 8-го порядка.

Plot[Evaluate[{Cos[x], Normal[Series[Cos[x], {x, 0, 8}]]}], {x, -2π, 2π}]



Чтобы узнать в каком виде хранится тот или иной объект в системе *Mathematica* можно использовать функцию `InputForm`. Мы уже знаем, что ряды Тейлора хранятся в виде `SeriesData` объекта.

InputForm[log1]

SeriesData[x, 0, {1, −1/2, 1/3, −1/4, 1/5, −1/6, 1/7}, 1, 8, 1]

Соответственно, функция `SeriesData[`x`,` x_0`, {`a_0`,` a_1`, …}, `n_{min}`,` n_{max}`, den]` создает объект `SeriesData`, который представляет степенной ряд в окрестности точки x_0. Величины a_i являются коэффициентами ряда.

SeriesData[x, 0, {1, −1/2, 1/3, −1/4, 1/5, −1/6, 1/7}, 1, 8, 1]

$$x-\frac{x^2}{2}+\frac{x^3}{3}-\frac{x^4}{4}+\frac{x^5}{5}-\frac{x^6}{6}+\frac{x^7}{7}+O[x]^8$$

SeriesData[x, 0, Table[$i*\sqrt{i}$, {i, 10}], 0, 7, 1]

$$1+2\sqrt{2}x+3\sqrt{3}x^2+8x^3+5\sqrt{5}x^4+6\sqrt{6}x^5+7\sqrt{7}x^6+O[x]^7$$

Ряд может быть построен не по целым степеням. Для этого используется последний аргумент `den`, отличный от единицы. В таком случае генерируется ряд с коэффициентами $\{a_0, a_1, \ldots\}$ при степенях $(x-x_0)^{n_{min}/den}$, $(x-x_0)^{(n_{min}+1)/den}$ и т.д. до $(x-x_0)^{n_{max}/den}$.

SeriesData[x, 0, {1, 1, 1/2, 1/6, 1/24, 1/120}, 0, 6, 2]

$$1+\sqrt{x}+\frac{x}{2}+\frac{x^{3/2}}{6}+\frac{x^2}{24}+\frac{x^{5/2}}{120}+O[x]^3$$

Функция `CoefficientList[poly,var]` возвращает список коэффициентов полинома `poly` при степенях переменной `var`.

CoefficientList[$(3+x)^5$, x]

```
{243,405,270,90,15,1}
```

CoefficientList[$(x-1)(x-2)(x-3)$, x]

```
{-6,11,-6,1}
```

CoefficientList[$1+x+xy+y^2-x^3+y^3$, {x, y}]//TableForm

1	0	1	1
1	1	0	0
0	0	0	0
−1	0	0	0

Эта фукция, в каком то смысле, является обратной к `SeriesData`.

se = Series[Exp[x], {x, 0, 5}]

CoefficientList[Normal[se], x]

$$\{1,1,\frac{1}{2},\frac{1}{6},\frac{1}{24},\frac{1}{120}\}$$

Функция `SeriesCoefficient[series,n]` возвращает коэффициент при $n-ом$ члене степенного ряда.

SeriesCoefficient[**log1**, **5**]

$$\frac{1}{5}$$

3.5.3 Ряды Фурье

Периодические функции с периодом $2 \cdot L$ можно разложить в ряд по синусам и косинусам

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left[a_n \cos(n\pi x / L) + b_n \sin(n\pi x / L) \right], \tag{1}$$

в котором коэффициенты a_n и b_n вычисляются по формулам

$$\begin{aligned} a_n &= \frac{1}{L} \int_{-L}^{L} f(x) \cos(n\pi x / L) dx, \ n = 0,1,2,\ldots, \\ b_n &= \frac{1}{L} \int_{-L}^{L} f(x) \sin(n\pi x / L) dx, \ n = 1,2,\ldots, \end{aligned} \tag{2}$$

В ряд Фурье можно разложить только периодическую функцию. Но если функция определена на конечном интервале, например на интервале $-L \le x \le L$, то ее всегда можно периодически с периодом $2 \cdot L$ продолжить на всю числовую ось и затем использовать разложение для периодической функции. Часто в качестве L выбирают число π. Ряды Фурье удобно записывать в комплексном виде

$$f(x) = \sum_{n=-\infty}^{\infty} c_n e^{i n \pi x / L}, \tag{3}$$

где

$$c_n = \frac{1}{2L} \int_{-L}^{L} f(x) e^{-i n \pi x / L} dx, \ n = 0, \pm 1, \pm 2, \ldots. \tag{4}$$

Заметим, что для рядов Фурье имеет место среднеквадратичная сходимость к функции, а не поточечная.

Разложение в ряд Фурье можно интерпретировать как представление функции `f(x)` последовательностью чисел d_n вида

$$d_n = \sqrt{a_n^2 + b_n^2} = |c_n| \quad n = 0,1,2,\ldots \tag{5}$$

где коэффициенты d_n служат мерой вклада соответствующих частотных компонент в функцию `f(x)`. Последовательность $\{d_n\}$ называют частотным спектром функции `f`.

В пакете *Mathematica* имеется несколько функций, выполняющих разложение в ряды Фурье.

Функция `FourierTrigSeries[expr,t,n]` раскладывает выражение `expr`, заданное на отрезке $[-\pi, \pi]$ и зависящее от переменной `t`, в тригонометрический ряд Фурье порядка `n`.

FourierTrigSeries$[t^2 - t, t, 3]$

$$\frac{\pi^2}{3} + 4\left(-\text{Cos}[t] + \frac{1}{4}\text{Cos}[2t] - \frac{1}{9}\text{Cos}[3t]\right) + 2\left(-\text{Sin}[t] + \frac{1}{2}\text{Sin}[2t] - \frac{1}{3}\text{Sin}[3t]\right)$$

FourierTrigSeries$[\text{Cos}[t]^2 + \text{Sin}[t]^3, t, 3]$

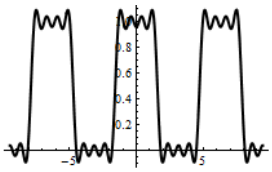
$$\frac{1}{2} + \frac{1}{2}\text{Cos}[2t] + \frac{3\text{Sin}[t]}{4} - \frac{1}{4}\text{Sin}[3t]$$

Точность приближения рядом Фурье разрывных функций хуже, чем непрерывных.

FourierTrigSeries$[$**UnitBox**$[t/\pi], t, 8]$

$$\frac{1}{2} + \frac{2\text{Cos}[t]}{\pi} - \frac{2\text{Cos}[3t]}{3\pi} + \frac{2\text{Cos}[5t]}{5\pi} - \frac{2\text{Cos}[7t]}{7\pi}$$

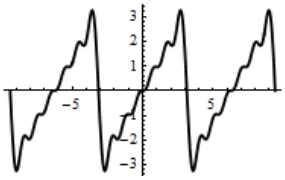
Plot$[\%, \{t, -3\pi, 3\pi\}]$



FourierTrigSeries$[t, t, 6]$

$$2\text{Sin}[t] - \text{Sin}[2t] + \frac{2}{3}\text{Sin}[3t] - \frac{1}{2}\text{Sin}[4t] + \frac{2}{5}\text{Sin}[5t] - \frac{1}{3}\text{Sin}[6t]$$

Plot$[\%, \{t, -3\pi, 3\pi\}]$

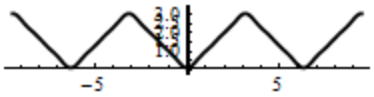


Чем выше гладкость функции, тем лучше приближение при одном и том же порядке ряда/суммы Фурье.

FourierTrigSeries$[$**Abs**$[t], t, 6]$

$$\frac{\pi}{2} - \frac{4\text{Cos}[t]}{\pi} - \frac{4\text{Cos}[3t]}{9\pi} - \frac{4\text{Cos}[5t]}{25\pi}$$

Plot$[\%, \{t, -3\pi, 3\pi\},$ **AspectRatio** $\rightarrow$ **Automatic**$]$

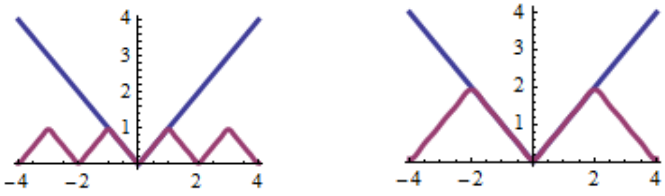


При установке опции `FourierParameters->{1,π/L}` создается ряд (1) с коэффициентами (2). Следующий ряд построен на отрезке [-1, 1] и имеет период 2 (следующий рисунок слева).

fts1$[$t_$] =$ **FourierTrigSeries**$[$**Abs**$[t], t, 6,$ **FourierParameters** $\rightarrow \{1, \pi\}]$

Plot$[\{$**Abs**$[t],$ **fts1**$[t]\}, \{t, -4, 4\},$ **PlotStyle** $\rightarrow$ **Thickness**$[0.01]]$

$$\frac{1}{2} - \frac{4\text{Cos}[\pi t]}{\pi^2} - \frac{4\text{Cos}[3\pi t]}{9\pi^2} - \frac{4\text{Cos}[5\pi t]}{25\pi^2}$$



Следующий ряд имеет период 4 (L=2).

$$\mathbf{fts2[t\_] = FourierTrigSeries[Abs[t], t, 6, FourierParameters \to \{1, \pi/2\}]}$$

$$\mathbf{Plot[\{Abs[t], fts2[t]\}, \{t, -4, 4\}, PlotStyle \to Thickness[0.02]]}$$

$$1 - \frac{8\text{Cos}[\frac{\pi t}{2}]}{\pi^2} - \frac{8\text{Cos}[\frac{3\pi t}{2}]}{9\pi^2} - \frac{8\text{Cos}[\frac{5\pi t}{2}]}{25\pi^2}$$

(предыдущий рисунок справа).

Опция `Assumptions` уточняет диапазон изменения символьных переменных, используемых в разлагаемой функции.

$$\mathbf{FourierTrigSeries[Abs[t - a], t, 3, Assumptions \to 0 < a < \pi]}$$

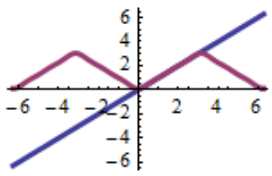
$$\frac{a^2+\pi^2}{2\pi} - \frac{2(1+\text{Cos}[a])\text{Cos}[t]}{\pi} - \frac{2(1+\text{Cos}[3a])\text{Cos}[3t]}{9\pi} + \frac{\text{Cos}[2t]\text{Sin}[a]^2}{\pi} - \frac{2(a+\text{Sin}[a])\text{Sin}[t]}{\pi} + \frac{(a-\text{Cos}[a]\text{Sin}[a])\text{Sin}[2t]}{\pi} - \frac{2(3a+\text{Sin}[3a])\text{Sin}[3t]}{9\pi}$$

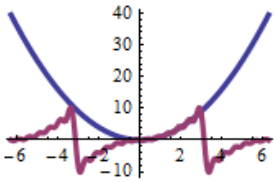
В разложении четных функций присутствуют только косинусы. Любую функцию, заданную на отрезке [0,L] можно продолжить четно на отрезок [-L,0], а потом разложить в ряд Фурье. Но можно сразу использовать формулы разложения по косинусам. Функция `FourierCosSeries` выполняет эту работу сразу (следующий рисунок слева).

$$\mathbf{fcs[t\_] = FourierCosSeries[t, t, 6]}$$

$$\mathbf{Plot[\{t, fcs[t]\}, \{t, -2\pi, 2\pi\}, PlotStyle \to Thickness[0.02]]}$$

$$\frac{\pi}{2} - \frac{4\text{Cos}[t]}{\pi} - \frac{4\text{Cos}[3t]}{9\pi} - \frac{4\text{Cos}[5t]}{25\pi}$$





Сравните с результатом одного из примеров, приведенных выше, где разлагалась функция Abs[t].

Косинус преобразование функции f(t) эквивалентно преобразованию Фурье функции $\begin{cases} f(t), t \ge 0, \\ f(-t), t < 0 \end{cases}$

В разложении нечетных функций присутствуют только синусы. Любую функцию, заданную на отрезке [0,L], можно нечетно продолжить на отрезок $[-L, 0]$, а потом разложить в ряд Фурье. Но можно сразу использовать формулы разложения по синусам. Функция `FourierSinSeries` выполняет эту работу сразу.

$$\mathbf{fss[t\_] = FourierSinSeries[t^2, t, 12];}$$

$$\mathbf{Plot[\{t^2, fss[t]\}, \{t, -2\pi, 2\pi\}, PlotStyle \to Thickness[0.02]]}$$

(предыдущий рисунок справа).

Синус преобразование функции $f(t)$ эквивалентно преобразованию Фурье функции $\begin{cases} f(t), t \ge 0, \\ -f(-t), t < 0 \end{cases}$.

Функция FourierSeries используется для построения комплексных рядов Фурье.

$$\mathbf{fs[t\_] = FourierSeries[}t, t, 3\mathbf{]}$$

$$ii e^{-it} - ii e^{it} - \frac{1}{2} ii e^{-2it} + \frac{1}{2} ii e^{2it} + \frac{1}{3} ii e^{-3it} - \frac{1}{3} ii e^{3it}$$

Легко видеть, что вещественная часть результата совпадает с результатом работы функции FourierTrigSeries, а мнимая равна нулю.

$$\mathbf{ComplexExpand[Re[fs[}t\mathbf{]]]}$$

$$2\text{Sin}[t] - \text{Sin}[2t] + \frac{2}{3}\text{Sin}[3t]$$

$$\mathbf{ComplexExpand[Im[fs[}t\mathbf{]]]}$$

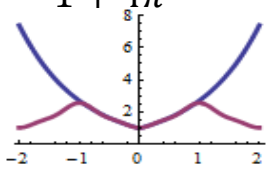
0

Опции функций FourierSeries, FourierTrigSeries, FourierSinSeries, FourierCosSeries, используются одинаково. Например,

$$\mathbf{fcs[t\_] = FourierCosSeries[Exp[}t\mathbf{], }t\mathbf{, 3, FourierParameters} \rightarrow \{1, \pi\}\mathbf{]}$$

$$\mathbf{Plot[\{Exp[Abs[}t\mathbf{]], fcs[}t\mathbf{]\}, \{}t, -2, 2\mathbf{\}, PlotStyle} \rightarrow \mathbf{Thickness[0.01]]}$$

$$-1 + e - \frac{2(1+e)\text{Cos}[\pi t]}{1+\pi^2} + \frac{2(-1+e)\text{Cos}[2\pi t]}{1+4\pi^2} - \frac{2(1+e)\text{Cos}[3\pi t]}{1+9\pi^2}$$



Здесь мы построили график четного продолжения функции e^t с положительной полуоси на отрицательную, т.е. график функции $e^{|t|}$; а также косинус разложение функции e^t при L=1 (для этого использовалась опция FourierParameters→{1,π}), которое приближает функцию $e^{|t|}$ на отрезке [-L,L].

Любую функцию этой группы можно использовать для построения ряда Фурье функции нескольких переменных.

$$\mathbf{FourierSeries[}x + y, \{x, y\}, \{2, 2\}\mathbf{]}$$

$$ii e^{-ix} - ii e^{ix} - \frac{1}{2} ii e^{-2ix} + \frac{1}{2} ii e^{2ix} + ii e^{-iy} - ii e^{iy} - \frac{1}{2} ii e^{-2iy} + \frac{1}{2} ii e^{2iy}$$

$$\mathbf{FourierTrigSeries[}x + y, \{x, y\}, \{2, 2\}\mathbf{]}$$

$$2\text{Sin}[x] - \text{Sin}[2x] + 2\text{Sin}[y] - \text{Sin}[2y]$$

$$\mathbf{FourierSinSeries[}x * y, \{x, y\}, \{2, 2\}\mathbf{]}$$

$$4\text{Sin}[x]\text{Sin}[y] - 2\text{Sin}[2x]\text{Sin}[y] - 2\text{Sin}[x]\text{Sin}[2y] + \text{Sin}[2x]\text{Sin}[2y]$$

$$\mathbf{FourierCosSeries[}x^2 + y^2, \{x, y\}, \{2, 2\}\mathbf{]}$$

$$\frac{8\pi^2}{3} - 8\text{Cos}[x] + 2\text{Cos}[2x] - 8\text{Cos}[y] + 2\text{Cos}[2y]$$

Функция FourierCoefficient возвращает n-й коэффициент в разложении функции в ряд Фурье, где n может быть любым целым числом. Так в формате FourierCoefficient[expr,t,k] возвращается коэффициент при e^{ikt}. Например,

```
fc[k_]:= FourierCoefficient[t, t, k]
Table[fc[k], {k, -3, 3}]
```

$$\left\{\frac{i}{3}, -\frac{i}{2}, i, 0, -i, \frac{i}{2}, -\frac{i}{3}\right\}$$

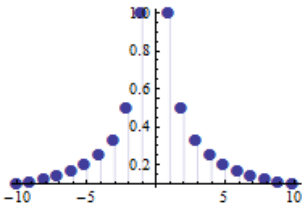
Или

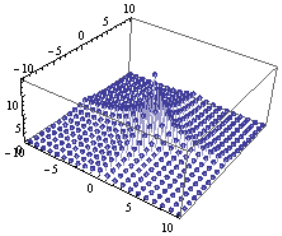
```
kn[n_] = FourierCoefficient[t, t, n]
```

$$\frac{i(-1)^n}{n}$$

Имея список коэффициентов или общую формулу для вычисления, можно построить частотный спектр функции (следующий рисунок слева).

```
DiscretePlot[Abs[kn[n]], {n, -10, 10}, PlotRange → All,
                PlotMarkers → {Automatic, Medium}]//Quiet
```





```
fc[m_, n_] = FourierCoefficient[Exp[-x - y], {x, y}, {m, n}]
```

$$-\frac{(-1)^{m+n}\,\mathrm{Sinh}[\pi]^2}{(-i+m)(-i+n)\pi^2}$$

```
ListPointPlot3D[Abs[Table[fc[m, n], {m, -10, 10}, {n, -10, 10}]],
   Filling → Bottom, PlotRange → All, DataRange → {{-10, 10}, {-10, 10}},
   PlotStyle → PointSize[Medium]]  (* предыдущий рисунок справа *)
```

Аналогично действуют функции `FourierSinCoefficient`, и `FourierCosCoefficient` .

```
FourierSinCoefficient[t, t, 3]
```

$$\frac{2}{3}$$

Это коэффициент при $\sin k\,x$. Действительно

```
Expand[FourierSinSeries[t, t, 4]]
```

$$2\mathrm{Sin}[t] - \mathrm{Sin}[2t] + \frac{2}{3}\mathrm{Sin}[3t] - \frac{1}{2}\mathrm{Sin}[4t]$$

Коэффициенты рядов Фурье вычисляются как интегралы по формулам (2) или (4). Вы можете сравнить.

```
FourierCoefficient[t, t, 3]
```

$$-\frac{i}{3}$$

Полученное число это коэффициент при e^{i3t} в разложении Фурье функции $f(t)=t$ на интервале $[-\pi,\pi]$. Тогда, в соответствии с (4), он равен

```
1/(2π)Integrate[t Exp[-3It], {t, -π, π}]
```

$$-\frac{i}{3}$$

3.6 Поиск экстремальных значений.

Многие задачи – как из области математики, так и из других областей науки и техники – приводят к задаче о нахождении наибольшего или наименьшего значения некоторой функции. В системе *Mathematica* имеется большой набор функций, предназначенных для решения таких задач.

3.6.1 Экстремумы дискретных наборов

Для поиска минимальных или максимальных значений среди чисел, входящих в список, используются функции `Min` и `Max`.

В формате `Min[x1,x2,...]` или `Max[x1,x2,...]` функция возвращает минимальный/максимальный элемент из набора `x1,x2,...`.

```
Min[5, 2, 14]
```
2

Функции работают как с приближенными, так и точными числами.

```
Min[5, 4.5, Sqrt[14]]
```
$\sqrt{14}$

В формате `Min[{x1,x2,...},{y1,y2,...},...]` функция возвращает минимальный элемент из всех списков. Аналогичный формат есть у функции `Max`.

```
Min[{4, 1, 7, 2}]
```
1

```
Min[{3, 4, 3}, {2, Sqrt[2]}, 7]
```
$\sqrt{2}$

```
M = {{-2, 0, 3, 2}, {0, 2, 7, 6}, {-6, -2, -4, 0}};
Min[M]
```
-6

Но эти функции можно использовать и для поиска минимума/максимума каждой строки или каждого столбца матрицы.

```
Min/@M
```
```
{-2,0,-6}
```
```
Min/@Transpose[M]
```
```
{-6,-2,-4,0}
```

Напомним, что операция `/@` является инфиксной формой функции `Map`, которая в формате `Map[f,expr]` применяет функцию `f` к каждому элементу первого уровня в выражении `expr`. В нашем примере функция `Min` проносится внутрь внешнего списка и применяется к каждому внутреннему списку матрицы M.

Функции можно использовать с символьными переменными

```
Min[x, y, Min[x, z]]
```
```
Min[x,y,z]
```

и выполнять алгебраические преобразования

```
FullSimplify[Max[x, y] - Abs[x - y]]
```

$$\begin{cases} x & x < y \\ y & \text{True} \end{cases}$$

FullSimplify[**Max**[x, y] − **Abs**[$x - y$] − **Min**[x, y]]

0

Функции `Min` и `Max` можно дифференцировать

$\boldsymbol{D}$[**Min**[x, x^2], x]

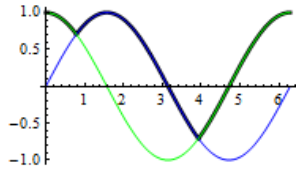
$$\begin{cases} 1 & x < 0 \\ 2x & 0 < x < 1 \\ 1 & x > 1 \\ \text{Indeterminate} & \text{True} \end{cases}$$

Функции `Min` и `Max` можно интегрировать.

Integrate[**Max**[**Sin**[x], **Cos**[x]], {x, **0**, **2Pi**}]

$2\sqrt{2}$

Plot[{**Max**[**Sin**[x], **Cos**[x]], **Sin**[x], **Cos**[x]}, {x, **0**, **2Pi**},
PlotRange → **All**, **PlotStyle** → {{**Black**, **Thickness**[**0.015**]}, **Blue**, **Green**}]



Функции `Min` и `Max` можно использовать в граничных значениях итераторов.

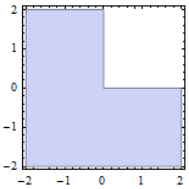
Table[$i + j + k$, {i, **2**}, {j, **3**}, {k, **Max**[i, j], **3**}]

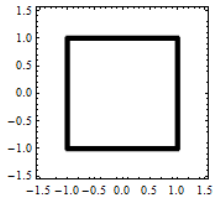
{{{3,4,5},{5,6},{7}},{{5,6},{6,7},{8}}}

Их можно использовать при построении графических объектов.

RegionPlot[**Min**[x, y] < 0, {x, **−2**, **2**}, {y, **−2**, **2**}] (*следующий рисунок слева *)

ContourPlot[**Max**[{**Abs**[x], **Abs**[y]}] == **1**, {x, **−1.5**, **1.5**}, {y, **−1.5**, **1.5**},
ContourStyle → {**Black**, **Thickness**[**0.03**]}] (*следующий рисунок справа *)

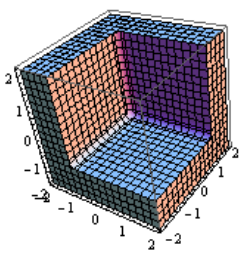


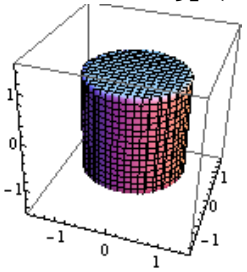


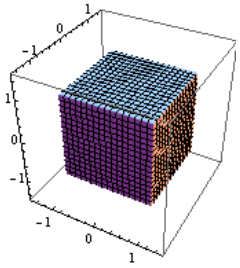
RegionPlot3D[**Min**[x, $-y$, z] < −1,
{x, **−2**, **2**}, {y, **−2**, **2**}, {z, **−2**, **2**}] (* следующий рисунок слева *)

RegionPlot3D[**Max**[$x^2 + y^2$, z^2] < 1, {x, **−1.5**, **1.5**}, {y, **−1.5**, **1.5**},
{z, **−1.5**, **1.5**}, **PlotPoints** → **30**] (* следующий рисунок в центре *)

ContourPlot3D[**Max**[{**Abs**[x], **Abs**[y], **Abs**[z]}] == **1**,
{x, **−1.5**, **1.5**}, {y, **−1.5**, **1.5**}, {z, **−1.5**, **1.5**}] (*следующий рисунок справа *)







Близкими по смыслу являются функции `RankedMin` и `RankedMax`. Они могут вычислить `n`-й наименьший или наибольший элемент списка.

$$\mathbf{RankedMin}[\{1, 15, 6, 21, 8, 12, 13, 11\}, 3] \quad \text{(* третий наименьший *)}$$

8

$$\mathbf{RankedMax}[\{\pi, \sqrt{2}, e\}, 2] \quad \text{(* второй наибольший *)}$$

e

$$\mathbf{RankedMin}[\{1 - x, x, 1 + x\}, 1]\mathbf{//PiecewiseExpand}$$

$$\begin{cases} 1 - x & x \geq \frac{1}{2} \\ x & \text{True} \end{cases}$$

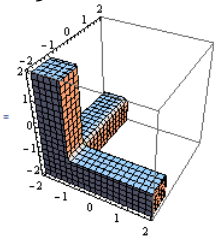
Это то же, что и

$$\mathbf{Min}[\{1 - x, x, 1 + x\}, 1]\mathbf{//PiecewiseExpand}$$

Здесь функция `PiecewiseExpand` преобразует вложенные кусочные (piecewise) функции в «одноуровневую» кусочную функцию.

Функции `RankedMin` и `RankedMax` также можно использовать при построении областей.

$$\mathbf{RegionPlot3D}[\mathbf{RankedMin}[\{x, y, z\}, 2] < -1,$$
$$\{x, -2, 2\}, \{y, -2, 2\}, \{z, -2, 2\}, \mathbf{PlotPoints} \to 19, \mathbf{Mesh} \to \mathbf{Full}]$$



3.6.2 Экстремумы функций

В системе *Mathematica* для поиска минимума или максимума функции, заданной аналитически, используются функции `Maximize` и `Minimize`. Они имеют одинаковые форматы вызова, поэтому для краткости мы будем описывать каждый формат только для одной из этих функций. Заметим, что эти функции заменили устаревшие функции `ConstrainedMin` и `ConstrainedMax` версии *Mathematica* 5.

В формате `Minimize[expr,x]` определяется минимум выражения `expr` по переменной *x*.

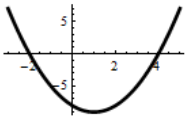
$$\mathbf{Minimize}[x^2 - 2x - 8, x]$$

$$\{-9, \{x \to 1\}\}$$

Результат возвращается в форме списка `{Мин_значение,{x→x`$_{min}$`}}`.

Вот график нашей функции.

$$\mathbf{Plot}[x^2 - 2x - 8, \{x, -3, 5\}]$$



Поскольку максимум функции $x^2 - 2x - 8$ равен бесконечности, то

$\mathbf{Maximize}[\boldsymbol{x^2 - 2x - 8}, \boldsymbol{x}]$

```
Maximize::natt: The maximum is not attained at any point
satisfying the given constraints.≫
```

$\{\infty, \{x \to -\infty\}\}$

и функция `Maximize` вывела сообщение.

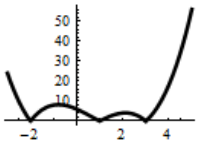
Обратите внимание, что первым аргументом функций `Maximize` и `Minimize` являются выражения, хотя по привычке мы говорим об экстремуме функции.

Исследуемые функции не обязаны быть гладкими.

$\boldsymbol{f}[\mathbf{x\_}] = \boldsymbol{x^3 - 2x^2 - 5x + 6}$;

$\mathbf{Plot}[\mathbf{Abs}[\boldsymbol{f}[\boldsymbol{x}]], \{\boldsymbol{x}, \boldsymbol{-3}, \boldsymbol{5}\}]$

$\mathbf{Minimize}[\mathbf{Abs}[\boldsymbol{f}[\boldsymbol{x}]], \boldsymbol{x}]$



$\{0, \{x \to -2\}\}$

Функции `Maximize` и `Minimize` умеют работать с символьными выражениями

$\mathbf{Minimize}[\boldsymbol{x^2 - 2ax}, \boldsymbol{x}]$

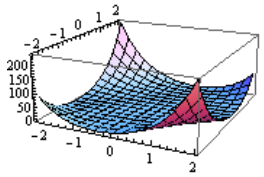
$\{-a^2, \{x \to a\}\}$

Исследуемые на экстремум выражения могут иметь любое конечное количество аргументов.

$\boldsymbol{f}[\mathbf{x\_}, \mathbf{y\_}] = (\boldsymbol{x^2 + y^2 + 1})((\boldsymbol{xy - 1})^2 + \boldsymbol{1})$

$\mathbf{Plot3D}[\boldsymbol{f}[\boldsymbol{x}, \boldsymbol{y}], \{\boldsymbol{x}, \boldsymbol{-2}, \boldsymbol{2}\}, \{\boldsymbol{y}, \boldsymbol{-2}, \boldsymbol{2}\}, \mathbf{PlotRange} \to \mathbf{All}]$

$\mathbf{Minimize}[\boldsymbol{f}[\boldsymbol{x}, \boldsymbol{y}], \{\boldsymbol{x}, \boldsymbol{y}\}]$

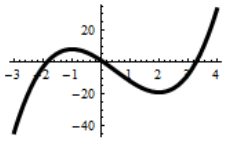


$\{2, \{x \to 0, y \to 0\}\}$

В формате `Minimize[{expr,ограничения},{x1,x2,...}]` определяется минимум выражения `expr` по переменным $x_1, x_2, \ldots$ с учетом `ограничений` (дополнительных условий).

$\boldsymbol{f}[\mathbf{x\_}] = \boldsymbol{2x^3 - 3x^2 - 12x + 1}$;

$\mathbf{Plot}[\boldsymbol{f}[\boldsymbol{x}], \{\boldsymbol{x}, \boldsymbol{-3}, \boldsymbol{5}\}]$



$\mathbf{Minimize}[\{\boldsymbol{f}[\boldsymbol{x}], \boldsymbol{-2 <= x \leq 3}\}, \boldsymbol{x}]$

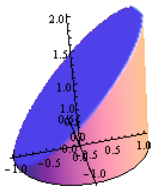
$\{-19, \{x \to 2\}\}$

$\mathbf{Maximize}[\{\boldsymbol{f}[\boldsymbol{x}], \boldsymbol{-2 <= x \leq 3}\}, \boldsymbol{x}]$

$\{8, \{x \to -1\}\}$

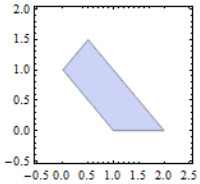
Вот пример для функции 2 – х переменных

$$\mathbf{Minimize}[\{\mathbf{1}+\boldsymbol{x},\boldsymbol{x}^{\mathbf{2}}+\boldsymbol{y}^{\mathbf{2}}\leq\mathbf{1}\},\{\boldsymbol{x},\boldsymbol{y}\}]$$
$$\mathbf{Maximize}[\{\mathbf{1}+\boldsymbol{x},\boldsymbol{x}^{\mathbf{2}}+\boldsymbol{y}^{\mathbf{2}}\leq\mathbf{1}\},\{\boldsymbol{x},\boldsymbol{y}\}]$$
$$\{0,\{x\to-1,y\to 0\}\}$$
$$\{2,\{x\to 1,y\to 0\}\}$$
$$\mathbf{RegionPlot3D}[\boldsymbol{x}^{\mathbf{2}}+\boldsymbol{y}^{\mathbf{2}}\leq\mathbf{1}\&\&\mathbf{0}\leq\boldsymbol{z}\leq\mathbf{1}+\boldsymbol{x},$$
$$\{\boldsymbol{x},-\mathbf{1},\mathbf{1}\},\{\boldsymbol{y},-\mathbf{1},\mathbf{1}\},\{\boldsymbol{z},\mathbf{0},\mathbf{2}\},\mathbf{PlotPoints}\to\mathbf{50},\mathbf{Mesh}\to\mathbf{None},$$
$$\mathbf{Boxed}\to\mathbf{False},\mathbf{AxesOrigin}\to\{\mathbf{0},\mathbf{0},\mathbf{0}\}]$$



Вот пример линейных ограничений и линейной целевой функции.

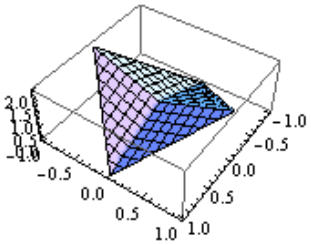
$$\mathbf{Maximize}[\{\boldsymbol{y},\boldsymbol{x}+\boldsymbol{y}-\mathbf{1}\geq\mathbf{0},-\boldsymbol{x}+\boldsymbol{y}-\mathbf{1}\leq\mathbf{0},\boldsymbol{x}+\boldsymbol{y}-\mathbf{2}<=0,\boldsymbol{y}\geq\mathbf{0}\},\{\boldsymbol{x},\boldsymbol{y}\}]$$
$$\{\frac{3}{2},\{x\to\frac{1}{2},y\to\frac{3}{2}\}\}$$
$$\mathbf{RegionPlot}[\boldsymbol{x}+\boldsymbol{y}-\mathbf{1}\geq\mathbf{0}\ \&\&\ -\boldsymbol{x}+\boldsymbol{y}-\mathbf{1}\leq\mathbf{0}\ \&\&\ \boldsymbol{x}+\boldsymbol{y}-\mathbf{2}\leq\mathbf{0}\ \&\&$$
$$\boldsymbol{y}\geq\mathbf{0},\{\boldsymbol{x},-\mathbf{0.5},\mathbf{2.5}\},\{\boldsymbol{y},-\mathbf{0.5},\mathbf{2}\}]$$



Здесь максимум достигается в самой высокой вершине многоугольника, представляющего множество точек, координаты которых удовлетворяют системе неравенств.

Целевая функция, зависящая от нескольких переменных, также не обязана быть гладкой.

$$\mathbf{Minimize}[\{\mathbf{1}-\mathbf{Abs}[\boldsymbol{x}]+\mathbf{Abs}[\boldsymbol{y}],\mathbf{Abs}[\boldsymbol{x}]+\mathbf{Abs}[\boldsymbol{y}]\leq\mathbf{1}\},\{\boldsymbol{x},\boldsymbol{y}\}]$$
$$\{0,\{x\to-1,y\to 0\}\}$$
$$\mathbf{Plot3D}[\mathbf{1}-\mathbf{Abs}[\boldsymbol{x}]+\mathbf{Abs}[\boldsymbol{y}],\{\boldsymbol{x},-\mathbf{1},\mathbf{1}\},\{\boldsymbol{y},-\mathbf{1},\mathbf{1}\},$$
$$\mathbf{RegionFunction}\to\mathbf{Function}[\{\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}\},\mathbf{Abs}[\boldsymbol{x}]+\mathbf{Abs}[\boldsymbol{y}]\leq\mathbf{1}]]$$



Целевая функция может быть неограничена. Тогда система может вернуть следующий результат

$$\mathbf{Minimize}[\{\boldsymbol{x}+\boldsymbol{y},\boldsymbol{xy}\leq\mathbf{1}\},\{\boldsymbol{x},\boldsymbol{y}\}]$$
```
Minimize::natt: The minimum is not attained at any point
satisfying the given constraints. ≫
```
$$\{-\infty,\{x\to\text{Indeterminate},y\to\text{Indeterminate}\}\}$$

Экстремум может существовать, но не достигаться в области. Тогда результат вычисления может выглядеть следующим образом

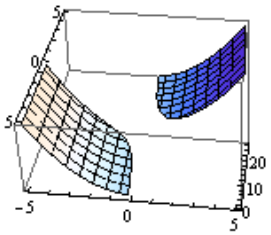
$$\mathbf{Minimize}[\{\mathbf{1}+\boldsymbol{x}^{\mathbf{2}},\boldsymbol{x}\,\boldsymbol{y}\geq\mathbf{1}\},\{\boldsymbol{x},\boldsymbol{y}\}]$$

```
Minimize::natt: The minimum is not attained at any point
satisfying the given constraints. ≫
```

$$\{1,\{x\to \text{Indeterminate},y\to \text{Indeterminate}\}\}$$

Следующий график целевой функции проясняет ситуацию.

$$\mathbf{Plot3D}[\mathbf{1}+\boldsymbol{x}^{\mathbf{2}},\{\boldsymbol{x},-\mathbf{5},\mathbf{5}\},\{\boldsymbol{y},-\mathbf{5},\mathbf{5}\},\mathbf{RegionFunction}\to\mathbf{Function}[\{\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}\},\boldsymbol{x}\boldsymbol{y}\geq\mathbf{1}]]$$



Глобальный минимум достигается при x=0, а условие $x\,y\geq 1$ исключает значение x=0 из рассмотрения, хотя при достаточно большом y величина x может приблизиться к нулю сколь угодно близко.

Если ограничение задает открытую область (без границы), а экстремум принадлежит границе, то функции `Maximize/Minimize` могут возвратить точку, которая лежит на границе и вывести подходящее сообщение.

$$\mathbf{Maximize}[\{\boldsymbol{x}\boldsymbol{y},\boldsymbol{x}^{\mathbf{2}}+\boldsymbol{y}^{\mathbf{2}}<1\},\{\boldsymbol{x},\boldsymbol{y}\}]$$

```
Maximize::wksol: Warning: there is no maximum in the region in
which the objective function is defined and the constraints are
satisfied; Mathematica will return a result on the boundary. ≫
```

$$\{\frac{1}{2},\{x\to-\frac{1}{\sqrt{2}},y\to-\frac{1}{\sqrt{2}}\}\}$$

Функции умеют решать задачи целочисленного программирования.

$$\mathbf{Maximize}[\{\mathbf{2}\boldsymbol{x}+\mathbf{3}\boldsymbol{y}^{\mathbf{2}}-\mathbf{2}\boldsymbol{z}^{\mathbf{3}},\mathbf{1}\leq\boldsymbol{x}+\boldsymbol{y}+\boldsymbol{z}\leq\mathbf{2}\ \&\&\ \mathbf{1}\leq\boldsymbol{x}-\boldsymbol{y}+\boldsymbol{z}\leq\mathbf{2}\ \&\&\ \boldsymbol{x}-\boldsymbol{y}-\boldsymbol{z}==\mathbf{3}\ \&\&\ \mathbf{Element}[\{\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}\},\mathbf{Integers}]\},\{\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}\}]$$

$$\{6,\{x\to 2,y\to 0,z\to -1\}\}$$

Здесь к ограничениям – неравенствам мы добавили дополнительное условие `Element[{x,y,z},Integers]`.

Ограничения могут содержать уравнения, неравенства, их логическую комбинацию и, возможно, ограничение «целочисленности». Если выражение `f` и ограничения являются линейными или полиномиальными функции `Minimize` и `Maximize` будут всегда искать глобальный (в области) экстремум. Если этим функциям переданы выражения, содержащие приближенные числа, то экстремум они будут искать численно (а не символьно), автоматически вызывая функции `NMinimize` и `NMaximize`. Если области «принадлежности» переменных не заданы, то полагается, что все переменные вещественные. Кроме того, следует помнить, что функции возвращают одно значение экстремума, даже если одинаковых экстремальных значений имеется несколько.

Естественно, что функции `Minimize` и `Maximize` успешно решают задачи линейного программирования.

$$\mathbf{Maximize}[\{x-3y-z, 1 \le x+y+2z \le 2 \;\&\&\; 1 \le x-3y+z \le 2 \;\&\&\; x-2y-5z == 3\}, \{x,y,z\}]$$

$$\{\frac{12}{5}, \{x \to \frac{8}{5}, y \to -\frac{1}{5}, z \to -\frac{1}{5}\}\}$$

Однако следует знать, что для таких задач в системе имеется специальная функция `LinearProgramming`, которая ожидает входные данные в матричном виде (в виде списков). Например,

$\boldsymbol{c} = \{1,-3,-1\}$; (*коэффициенты целевой функции*)

$\boldsymbol{m} = \{\{1,1,2\},\{1,1,2\},\{1,-3,1\},\{1,-3,1\},\{1,-2,-5\}\}$;(*коэф. ограничений*)

$\boldsymbol{b} = \{\{1,1\},\{2,-1\},\{1,1\},\{2,-1\},\{3,0\}\}$; (*знач. прав. части, тип неравенства*)

$$\mathbf{LinearProgramming}[-\boldsymbol{c}, \boldsymbol{m}, \boldsymbol{b}, -\mathbf{Infinity}]$$

$$\{\frac{8}{5}, -\frac{1}{5}, -\frac{1}{5}\}$$

Само значение целевой функции находим следующим образом

$\boldsymbol{c}.\%$

$$\frac{12}{5}$$

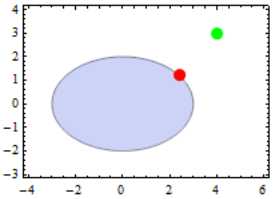
Подробнее с функцией `LinearProgramming` вы можете познакомиться по справочной системе.

Пример. Найдем минимальное расстояние от точки (4,3) до эллипса $\frac{x^2}{9}+\frac{y^2}{4}=1$. Имеем

$$\mathbf{m} = \mathbf{Minimize}[\{(\mathbf{x}-4)^2+(\mathbf{y}-3)^2, \frac{\mathbf{x}^2}{9}+\frac{\mathbf{y}^2}{4} \le 1\}, \{\mathbf{x},\mathbf{y}\}]$$

$$\{\frac{29}{5}, \{x \to \frac{12}{5}, y \to \frac{6}{5}\}\}$$

$$\mathbf{RegionPlot}[\frac{x^2}{9}+\frac{y^2}{4} \le 1, \{x,-4,6\}, \{y,-3,4\},$$
$$\mathbf{Epilog} \to \{\mathbf{PointSize}[0.05], \{\mathbf{Red}, \mathbf{Point}[\{x,y\}/.m[[2]]]\},$$
$$\{\mathbf{Green}, \mathbf{Point}[\{4,3\}]\}\}, \mathbf{AspectRatio} \to \mathbf{Automatic}]$$



Пример. Среди всех прямоугольников единичного периметра найти прямоугольник с максимальной площадью. Обозначив через x и y длины сторон прямоугольника, получаем

$$\mathbf{Maximize}[\{x\,y,\; 2x+2y == 1 \;\&\&\; x > 0 \;\&\&\; y > 0\}, \{x,y\}]$$

$$\{\frac{1}{16}, \{x \to \frac{1}{4}, y \to \frac{1}{4}\}\}$$

Пример. Среди всех прямоугольных треугольников единичной площади найти треугольник с минимальным периметром. Обозначив через x и y длины сторон прямоугольника, имеем $S = \frac{1}{2}xy$, $P = x + y + \sqrt{x^2 + y^2}$. Тогда

$\mathbf{Minimize}[\{\boldsymbol{x} + \boldsymbol{y} + \sqrt{\boldsymbol{x^2} + \boldsymbol{y^2}}, \boldsymbol{x\,y} == \mathbf{2}\ \&\&\ \boldsymbol{x} > 0\ \&\&\ \boldsymbol{y} > 0\}, \{\boldsymbol{x}, \boldsymbol{y}\}]$

$\{2(1 + \sqrt{2}), \{x \to \cdots, y \to \cdots\}\}$

$\boldsymbol{N}[\%]$

```
{4.8284271,{x→1.4142136,y→1.4142136}}
```

Пример. Найти расстояние от заданной точки до параболы.

$\mathbf{x0 = 3; y0 = 3; Minimize}[\{\mathbf{Sqrt}[(\boldsymbol{x} - \mathbf{x0})^2 + (\boldsymbol{y} - \mathbf{y0})^2], \boldsymbol{y} == \boldsymbol{x^2}\}, \{\boldsymbol{x}, \boldsymbol{y}\}]$

$\{\frac{1}{2}\sqrt{43 - 14\sqrt{7}}, \{x \to \frac{1}{2}(1 + \sqrt{7}), y \to \frac{1}{4}(1 + \sqrt{7})^2\}\}$

$\mathbf{x0 = 3; y0 = 6; Minimize}[\{\mathbf{Sqrt}[(\boldsymbol{x} - \mathbf{x0})^2 + (\boldsymbol{y} - \mathbf{y0})^2], \boldsymbol{y} == \boldsymbol{x^2}\}, \{\boldsymbol{x}, \boldsymbol{y}\}]$

$\{\sqrt{\text{Root}[-6057 + 21157\#1 - 1192\#1^2 + 16\#1^3\&, 1]},$
$\{x \to \text{Root}[-3 - 11\#1 + 2\#1^3\&, 3], y \to \text{Root}[-3 - 11\#1 + 2\#1^3\&, 3]^2\}\}$

$\boldsymbol{N}[\%]$

```
{0.53948303,{x→2.4712314,y→6.1069848}}
```

Вы можете аналогично находить расстояние от точки до любой кривой, заменив уравнение кривой.

Функции `MinValue`, `ArgMin`, `MaxValue` и `ArgMax` решают те же задачи, которые решают функции `Maximize` и `Minimize`. Функции `MinValue` и `MaxValue` возвращают значение минимума и максимума, а функции `ArgMin`, и `ArgMax` возвращают координаты точки, в которой экстремум достигается. Фактически функция `MinValue[...]` эквивалентна команде `First[Minimize[...]]`, а функция `ArgMin[...]` эквивалентна команде `{x,y,...}/.Last[Minimize[...,{x,y,...},...]`. Аналогичное замечание касается функций `MaxValue` и `ArgMax`.

$\mathbf{MinValue}[\{(\boldsymbol{x} + \boldsymbol{y} - \mathbf{1})^2 + \mathbf{1}, \boldsymbol{x^2} + \boldsymbol{y^2} \leq \mathbf{1}\}, \{\boldsymbol{x}, \boldsymbol{y}\}]$

1

$\mathbf{ArgMin}[\{(\boldsymbol{x} + \boldsymbol{y} - \mathbf{1})^2 + \mathbf{1}, \boldsymbol{x^2} + \boldsymbol{y^2} \leq \mathbf{1}\}, \{\boldsymbol{x}, \boldsymbol{y}\}]$

$\{\frac{1}{2}, \frac{1}{2}\}$

$\mathbf{Minimize}[\{(\boldsymbol{x} + \boldsymbol{y} - \mathbf{1})^2 + \mathbf{1}, \boldsymbol{x^2} + \boldsymbol{y^2} \leq \mathbf{1}\}, \{\boldsymbol{x}, \boldsymbol{y}\}]$

$\{1, \{x \to \frac{1}{2}, y \to \frac{1}{2}\}\}$

Функции `NMaximize`, `NMinimize` численно находят максимум/минимум функции/выражения. Они возвращают результат в такой же форме как и функции `Maximize` и `Minimize`. Их также можно применять для решения задач целочисленного программирования.

Для линейной целевой функции и линейных ограничений эти функции определяют глобальный экстремум. Для других типов выражений только

гарантируется, что будет определен локальный экстремум, хотя функции все же пытаются найти глобальное значение.

В формате `NMinimize[expr,x]` численно определяется минимум выражения `expr` по переменной *x*. Аналогичный формат имеется у функции `NMaximize`.

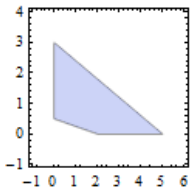
В формате `NMinimize[expr,{x1,x2,...}]` численно определяется минимум выражения `expr` по переменным $x_1, x_2, \ldots$. Аналогичный формат имеется у функции `NMaximize`.

В формате `NMinimize[{expr,ограничения},{x1,x2,...}]` численно определяется минимум выражения `expr` по переменным $x_1, x_2, \ldots$ с учетом `ограничений` (дополнительных условий). То же имеет место для функции `NMaximize`.

$$\mathbf{NMaximize}[-x^6 - 3x^4 + x, x]$$
```
{0.32123215,{x→0.42441332}}
```
$$\mathbf{NMaximize}[\{x^4 - (y-1)^4, x^2 + y^2 \leq 2\}, \{x, y\}]$$
```
{3.4397099,{x→1.3949862,y→0.23240812}}
```
$$\mathbf{NMaximize}[\{\mathbf{Exp}[\mathbf{Sin}[x]] - \mathbf{Sin}[10(x+y)] + (x^2+y^2)/4, x^2 + y^2 \leq 1\}, \{x, y\}]$$
```
{3.442848,{x→0.90372233,y→-0.42811908}}
```
В системе ограничений допустимо использовать операцию || (или).

$$\mathbf{NMaximize}[\{x + y, x^2 + y^2 \leq 1 || (x-1)^2 + y^2 \leq 1\}, \{x, y\}]$$
```
{2.4142136,{x→1.7071065,y→0.70710707}}
```
Можно потребовать, чтобы некоторые из искомых переменных были целыми числами.

$$\mathbf{NMaximize}[\{-x - 3y, 3x + 5y \leq 15 \;\&\&\; x + 4y \geq 2 \;\&\&\; x \geq 0 \;\&\&\; y \geq 0 \;\&\&\; \{x, y\} \in \mathbf{Integers}\}, \{x, y\}]$$
$$\{-2., \{x \to 2, y \to 0\}\}$$
$$\mathbf{NMaximize}[\{-x - 3y, 3x + 5y \leq 15 \;\&\&\; x + 4y \geq 2 \;\&\&\; x \geq 0 \;\&\&\; y \geq 0 \;\&\&\; \{x\} \in \mathbf{Integers}\}, \{x, y\}]$$
```
{-1.5,{x→0,y→0.5}}
```
$$\mathbf{RegionPlot}[3x + 5y \leq 15 \;\&\&\; x + 4y \geq 2 \;\&\&\; x \geq 0 \;\&\&\; y \geq 0, \{x, -1, 6\}, \{y, -1, 4\}]$$



В некоторых случаях функции `NMaximize` и `NMinimize` могут заменять функции `Solve` и `NSolve`. В следующем примере функция `NMaximize` находит единственный вещественный корень уравнения

$$\mathbf{NMaximize}[\{1, x^3 + x + 1 == 0\}, \{x\}]$$
```
{1.,{x→-0.68232781}}
```
Сравните

$$\mathbf{NSolve}[x^3 + x + 1 == 0, x]$$
$$\{\{x \to -0.68232781\},$$

$\{x \to 0.3411639 - 1.1615414i\},$
$\{x \to 0.3411639 + 1.1615414i\}\}$

Для функций `NMaximize` и `NMinimize` характерно использование опций, которые уместно использовать в численных алгоритмах.

Опция `WorkingPrecision` задает количество значащих цифр, используемых во внутренних вычислениях; установка `WorkingPrecision->MachinePrecision` (по умолчанию) приводит к вычислениям с процессорной точностью.

Опция `AccuracyGoal->n` задает количество значащих цифр абсолютной погрешности, которая считается равной 10^{-n}. Опция `PrecisionGoal->m` определяет «относительную погрешность» $|x|\cdot 10^{-m}$ вычислений и задается количеством значащих цифр. Установкой по умолчанию для этих опций является `WorkingPrecision/2`. Например, команда

NMinimize[Sin[*x*], *x*, AccuracyGoal → 12, PrecisionGoal → 8]

```
{-1.,{x→-1.5707963}}
```

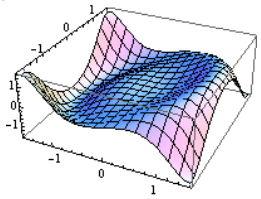
определяет следующий критерий остановки вычислений (погрешность) $\left\| x - x^* \right\| \le \max\left(10^{-12}, 10^{-8}\, x\right)$ и $\nabla f(x) \le 10^{-12}$. Первое неравенство относится к координатам точки экстремума, второе – к значению целевой функции.

Опцией `EvaluationMonitor` задается набор команд, которые вычисляются каждый раз, когда вычисляется значение выражения, стоящего в функции `NMinimize`.

Пример. Найдем максимальное значение функции $f(x,y) = x\sin\left(x^2 + y^2\right)$ в области $-\pi/2 \le x \le \pi/2$, $-\pi/2 \le y \le \pi/2$. Вначале построим график функции.

f[x_, y_] := x Sin[x^2 + y^2]

Plot3D[f[x, y], {x, -π/2, π/2}, {y, -π/2, π/2}, PlotRange → All]



Мы видим, что есть экстремумы в угловых точках области и локальные экстремумы внутри. Найдем их.

Maximize[{f[x, y], -π/2 ≤ x <= π/2, -π/2 ≤ y <= π/2}, {x, y}]

$$\{-\frac{1}{2}\pi \text{Sin}[\frac{1}{4}(6-\pi)\pi + \frac{\pi^2}{4}], \{x \to -\frac{\pi}{2}, y \to -\frac{1}{2}\sqrt{(6-\pi)\pi}\}\}$$

N[%]

```
{1.5707963,{x→-1.5707963,y→-1.4983284}}
```

Функция `Maximize` нашла максимум в области (один из двух имеющихся). Теперь посмотрим, что даст функция `NMaximize`.

NMaximize[{f[x, y], -π/2 ≤ x <= π/2, -π/2 ≤ y <= π/2}, {x, y}]

```
{1.3076194,{x→1.3552111,y→3.010915·10^-9}}
```

Функция `NMaximize` нашла локальный максимум внутри области.

Изобразим последовательность «пробных» точек, которые использовала функция `NMaximize` при приближении к локальному максимуму. Вначале запомним координаты «пробных» точек, которые расположены недалеко от экстремума. Для этого можно использовать опцию `EvaluationMonitor`.

```
{sol, pts} = Reap[NMaximize[{f[x, y], -π/2 ≤ x ≤ π/2, -π/2 ≤ y ≤ π/2},
                  {x, y}, EvaluationMonitor :→ Sow[{x, y}]]];
```

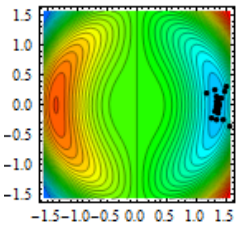
Напомним, что значение опции `EvaluationMonitor` является выражение, которое вычисляется каждый раз, когда вычисляется значение целевой функции. Функция `Sow` «выбрасывает в окружающее пространство» результат вычисления своего аргумента (в нашем случае пару {*x*,*y*}, содержащую координаты точки). Эти пары из «окружающего пространства подбирает» ближайшая объемлющая функция `Reap`. Первым элементом она возвращает результат вычисления основного выражения (в нашем случае результат помещается в переменную **sol**). Вторым элементом, который она возвращает, является список «собранных» ею пар {*x*,*y*} координат «пробных» точек. Можете посмотреть, чем являются переменные **sol** и **pts**.

```
sol
{1.3076194,{x→1.3552111,y→3.010915·10^-9}}
pts
{{{1.4894175,1.0840092},{-1.0709764,-0.44463881},...}
```

Мы не стали здесь приводить длинный список координат «пробных» точек.

Теперь, имея координаты «пробных» точек, выделим и изобразим только те, которые находятся рядом с экстремумом.

```
ContourPlot[f[x, y], {x, -π/2, π/2}, {y, -π/2, π/2},
    ColorFunction → (Hue[0.6#]&), Contours → 30,
    Epilog → {Black, PointSize[0.03], Map[Point,
                Cases[First[pts], z_/; Abs[f@@z - First[sol]] ≤ 0.5]]}]
```



Здесь выражение First[pts] имееет вид

```
First[pts]
{{1.4894175,1.0840092},{-1.0709764,-0.44463881},...}
```

Функция `Cases[{e1,e2,...},шаблон]` возвращает список элементов e_i, которые соответствуют шаблону. В нашем случае элементами являются пары чисел $\{x_i,y_i\}$. В результате шаблон z_ представляет список таких пар (т.е. он имеет вид List[x_i,y_i]). Условная операция /; (слэш – точка с запятой) выделяет из всех пар $\{x_i,y_i\}$ только те, которые удовлетворяют условию, записанному после операции. В нашем случае выделяются только те точки/списки $\{x_i,y_i\}$, в которых значение целевой функции f[x_i,y_i] находится близко к найденному экстремальному значению First[sol]. Само значение f[x_i,y_i] получается

использованием операции @@ (функции Apply), которая в нашем случае действует следующим образом: f@@z=Apply[f,{xi,yi}]= f[xi,yi].

Напомним, что функция `Apply[f,expr]` заменяет именем `f` заголовок выражения `expr` (в нашем случае `List`). С другими функциями, использованными в коде, мы знакомились ранее.

В функциях `NMaximize` и `NMinimize` можно использовать несколько разных алгоритмов численного поиска экстремума, которые можно задать с помощью опции `Method`. Иногда выбор одного из них, отличного от выбора по умолчанию, может дать лучший результат. Мы не ставим перед собой задачу изложения этих методов. С ними вы можете познакомиться по справочной системе.

У функций `NMaximize` и `NMinimize` есть опция `MaxIterations` , само название которой говорит о ее назначении, и опция `StepMonitor`, с которой мы познакомимся позже.

Для поиска локального минимума/максимума функции $f(x)$, в окрестности значения $x = x_0$, используются функции `FindMinimum` и `FindMaximum`. Они имеют идентичные форматы вызова, поэтому для краткости мы будем описывать форматы только одной из них. Результат возвращается в виде таком же, в каком возвращают результат функции `NMaximize` и `NMinimize`.

В формате `FindMinimum[f[x],{x,x0}]` численно ищется локальный минимум выражения `f[x]`; начальное значение задается точкой x_0. В формате `FindMinimum[f[x],x]` численно ищется локальный минимум выражения `f[x]`, начиная поиск из случайно выбираемой точки.

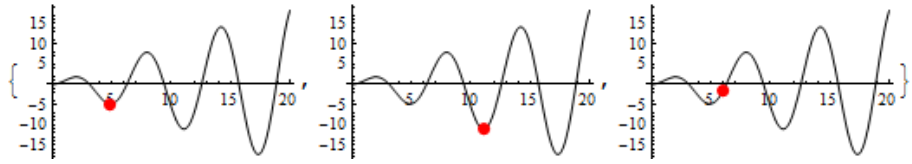
В формате `FindMinimum[f[x,y,...],{{x,x0},{y,y0},...}]` численно ищется локальный минимум функции нескольких переменных в окрестности точки $\{x_0, y_0, \ldots\}$. В формате `FindMinimum[{f[x,y,...],ограничения},{{x,x0},{y,y0},...}]` численно ищется локальный минимум с учетом `ограничений`. При этом точку $\{x_0, y_0, \ldots\}$ можно не задавать – она будет автоматически выбираться в области, определяемой ограничениями.

В следующем коде мы вызываем функцию `FindMinimum` три раза с разными начальными приближениями. При этом в третьем вызове мы задаем еще интервал изменения независимой переменной. Положение минимума, каждый раз разное, показано на графиках.

```
f[x_] = x Sin[x];
s1 = FindMinimum[f[x], {x, 3}]
s2 = FindMinimum[f[x], {x, 10}]
s3 = FindMinimum[{f[x], 6 ≤ x ≤ 15}, {x, 7}]
gplot[ss_] := Plot[f[x], {x, 0, 20}, Epilog → {xr = x /. ss[[2]];
                                {Red, PointSize[0.05], Point[{xr, f[xr]}]}}];
```

{gplot[s1], gplot[s2], gplot[s3]}

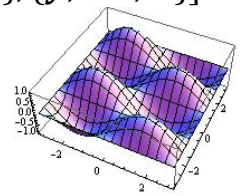
```
{-4.8144699,{x→4.9131804}}
{-11.040708,{x→11.085538}}
{-1.6764929,{x→6.}}
```



Как видим, выбор начальной точки влияет на положение точки минимума. В третьем вызове ближайший к начальной точке минимум оказался на границе области.

Аналогично для функции многих переменных, стартуя из разных точек, получаем разные локальные экстремумы. Рассмотрим функцию график которой показан на следующем рисунке.

Plot3D[Sin[x]Sin[2y], {x, $-\pi$, π}, {y, $-\pi$, π}]



Вызовем функцию `FindMinimum` с разными начальными точками.

FindMinimum[Sin[x]Sin[2y], {{x, 2}, {y, 2}}]

```
{-1.,{x→1.5707963,y→2.3561945}}
```

FindMinimum[Sin[x]Sin[2y], {{x, 0}, {y, 1}}]

```
{-1.,{x→-1.5707963,y→0.78539816}}
```

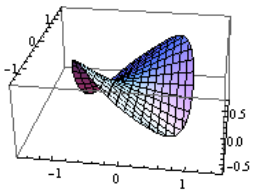
При задании области ограничений переменных начальную точку задавать не обязательно

FindMinimum[{Sin[x]Sin[2y], $x^2 + y^2 \leq 1$}, {x, y}]

```
{-0.66871014,{x→-0.79512351,y→0.60644782}}
```

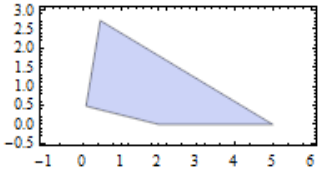
На следующем рисунке показана целевая функция в области единичного круга, который использовался при поиске экстремума.

Plot3D[Sin[x]Sin[2y], {x, -1.5, 1.5}, {y, -1.5, 1.5},
RegionFunction → Function[{x, y, z}, $x^2 + y^2 \leq 1$]]



Ограничения могут содержать условие «целочисленности» некоторых (или всех) переменных. Зададим следующую область

RegionPlot[$3x + 5y \leq 15$ && $x + 4y \geq 2$ && $y \leq 6x$ && $y \geq 0$,
{x, -1, 6}, {y, -0.5, 3}, AspectRatio → Automatic]



Теперь найдем минимум функции *x*+*y* в этой области, задавая условие «целочисленности» переменной *y*, и не задавая его.

$$\mathbf{FindMinimum}[\{\boldsymbol{x}+\boldsymbol{y}, \mathbf{3}\boldsymbol{x}+\mathbf{2}\boldsymbol{y} \le \mathbf{7} \;\&\&\; \boldsymbol{x}+\mathbf{2}\boldsymbol{y} \ge \mathbf{6} \;\&\&\; \boldsymbol{y} \le \mathbf{10}\boldsymbol{x} \;\&\&\; \boldsymbol{y} \ge \mathbf{0} \;\&\&\; \boldsymbol{y} \in \mathbf{Integers}\}, \{\boldsymbol{x}, \boldsymbol{y}\}]$$

```
{3.3,{x→0.3,y→3}}
```

$$\mathbf{FindMinimum}[\{\boldsymbol{x}+\boldsymbol{y}, \mathbf{3}\boldsymbol{x}+\mathbf{2}\boldsymbol{y} \le \mathbf{7} \;\&\&\; \boldsymbol{x}+\mathbf{2}\boldsymbol{y} \ge \mathbf{6} \;\&\&\; \boldsymbol{y} \le \mathbf{10}\,\boldsymbol{x} \;\&\&\; \boldsymbol{y} \ge \mathbf{0}\}, \{\boldsymbol{x}, \boldsymbol{y}\}]$$

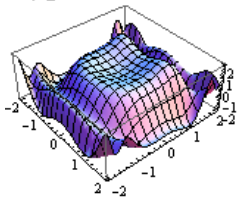
```
{3.1428571,{x→0.28571429,y→2.8571429}}
```

Естественно, что результат оказался разный.

Для функций `FindMinimum` и `FindMaximum` характерно использование опций, которые уместно использовать в численных алгоритмах. Опции `WorkingPrecision`, `AccuracyGoal`, `PrecisionGoal` используются точно также, как было описано для функций `NMaximize` и `NMinimize`. Опция `Method` имеет другие значения, поскольку алгоритмы поиска локального экстремума отличаются от алгоритмов поиска глобального. Новым является добавление опции `Gradient`. Она задает выражение для вычисления градиента минимизируемой / максимизируемой функции.

$$\boldsymbol{f}[\mathrm{x\_}, \mathrm{y\_}] := \mathbf{Cos}[\boldsymbol{x}^2 - \boldsymbol{y}^3] + \mathbf{Sin}[\boldsymbol{x}^2 + \boldsymbol{y}^2];$$

$$\mathbf{Plot3D}[\boldsymbol{f}[\boldsymbol{x}, \boldsymbol{y}], \{\boldsymbol{x}, -\mathbf{2}, \mathbf{2}\}, \{\boldsymbol{y}, -\mathbf{2}, \mathbf{2}\}]$$



$$\mathbf{FindMinimum}[\boldsymbol{f}[\boldsymbol{x}, \boldsymbol{y}], \{\boldsymbol{x}, \boldsymbol{y}\}, \mathbf{Gradient} :\to \{\mathbf{2}\boldsymbol{x}\mathbf{Cos}[\boldsymbol{x}^2 + \boldsymbol{y}^2] - \mathbf{2}\boldsymbol{x}\mathbf{Sin}[\boldsymbol{x}^2 - \boldsymbol{y}^3], \mathbf{2}\boldsymbol{y}\mathbf{Cos}[\boldsymbol{x}^2 + \boldsymbol{y}^2] + \mathbf{3}\boldsymbol{y}^2\mathbf{Sin}[\boldsymbol{x}^2 - \boldsymbol{y}^3]\}]$$

```
{-2.,{x→1.3446341,y→1.7042148}}
```

Опция `StepMonitor` позволяет контролировать шаги вычислений.

$$\boldsymbol{f}[\mathrm{x\_}] = \boldsymbol{x}^2 + \mathbf{1}/\sqrt{\boldsymbol{x}} - \mathbf{1};$$

$$\mathbf{FindMinimum}[\boldsymbol{f}[\boldsymbol{x}], \{\boldsymbol{x}, \mathbf{0.7}\}, \mathbf{StepMonitor} :\to \mathbf{Print}["\mathbf{x} = ", \boldsymbol{x}]]$$

```
x = 0.57904634
x = 0.573647
x = 0.57435347
x = 0.57434918
{0.64938489,{x→0.57434918}}
```

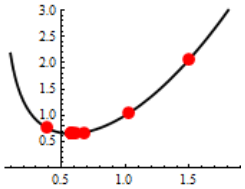
Можно, например, контролировать количество итераций

$$\boldsymbol{c} = \mathbf{0}; \{\mathbf{FindMinimum}[\boldsymbol{f}[\boldsymbol{x}], \{\boldsymbol{x}, \mathbf{0.7}\}, \mathbf{StepMonitor} :\to \boldsymbol{c}++], \boldsymbol{c}\}$$

```
{{0.64938489,{x→0.57434918}},4}
```

Используя эту опцию, можно изобразить промежуточные точки на каждом шаге вычислений.

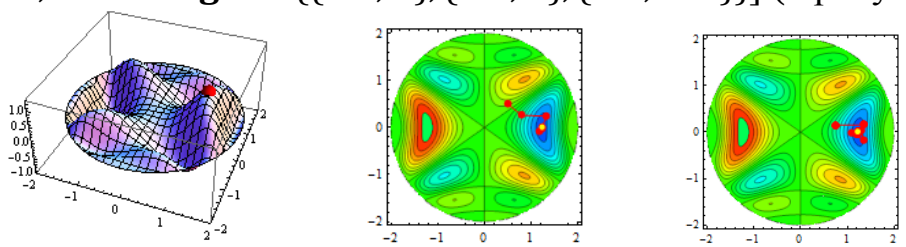
```
{sol, pts} = Reap[FindMinimum[f[x], {x, 3},
                    StepMonitor :→ Sow[{x, f[x]}]]]
Plot[f[x], {x, 0.1, 2}, Epilog → {Red, PointSize[0.05],
                    Map[Point, pts[[1]]]}, PlotRange → {0, 3}]
```



У опций `EvaluationMonitor` и `StepMonitor` есть различие в моменте вызова. Первая вызывается после каждого вычисления значения входной функции, вторая – после каждого шага итераций.

Пример. Изобразим путь, по которому двигаются точки алгоритма, приближая максимум функции $f(x,y)=\sin\left(x^2-y^2\right)\left(4-x^2-y^2\right)/3$ в области $x^2+y^2\le 4$. На следующем рисунке слева показана график функции в области ограничений и точка экстремума, который нашла функция `FindMinimum`.

```
f[x_, y_] := x Sin[x^2 - 2y^2](4 - x^2 - y^2)/3;
ps = Plot3D[f[x, y], {x, -2, 2}, {y, -2, 2},
    PlotRange → {{-2, 2}, {-2, 2}, {-1, 1}}, BoxRatios → {4, 4, 2},
    RegionFunction → Function[{x, y, z}, x^2 + y^2 ≤ 4], Mesh → 41];
{sol, pts} = Reap[FindMaximum[{f[x, y], x^2 + y^2 ≤ 4},
        {{x, 0.5}, {y, 0.5}}, StepMonitor :→ Sow[{x, y}]]];
{xp, yp} = {x, y}/. sol[[2]]; zp = sol[[1]];
pm = Graphics3D[{Red, Sphere[{xp, yp, zp}, 0.15]}];
Show[ps, pm, PlotRange → {{-2, 2}, {-2, 2}, {-1, 1.1}}] (* рисунок слева *)
```



На рисунке в середине показан контурный график исследуемой функции и ломаная, вершины которой представляют точки последовательного приближения. Желтая точка показывает положение максимума, который определила функция `FindMaximum`.

```
ContourPlot[f[x, y], {x, -2, 2}, {y, -2, 2}, ColorFunction → (Hue[0.6#]&),
    Contours → 21, RegionFunction → Function[{x, y, z}, x^2 + y^2 ≤ 4],
    Epilog → {Red, Line[pts], PointSize[0.04], Point[pts], Yellow,
            PointSize[0.03], Point[{xp, yp}]}] (* пред. рисунок в центре *)
```

Для сравнения изобразим ломаную, по которой функция `NMaximize` приближается к максимуму.

$$\{\mathbf{sol2}, \mathbf{pts2}\} = \mathbf{Reap}[\mathbf{NMaximize}[\{f[x,y], x^2 + y^2 \leq 4\}, \{x, y\},$$
$$\mathbf{StepMonitor} :\to \mathbf{Sow}[\{x, y\}]]];$$
$$\{\mathbf{xp2}, \mathbf{yp2}\} = \{x, y\}/.\,\mathbf{sol2}[[2]];$$

$$\mathbf{ContourPlot}[f[x,y], \{x, -2, 2\}, \{y, -2, 2\},$$
$$\mathbf{ColorFunction} \to (\mathbf{Hue}[0.6\#]\&), \mathbf{Contours} \to 21,$$
$$\mathbf{RegionFunction} \to \mathbf{Function}\left[\{x, y, z\}, x^2 + y^2 \leq 4\right],$$
$$\mathbf{Epilog} \to \{\mathbf{Red}, \mathbf{Line}[\mathbf{pts2}], \mathbf{PointSize}[0.04], \mathbf{Point}[\mathbf{pts2}],$$
$$\mathbf{Yellow}, \mathbf{PointSize}[0.03], \mathbf{Point}[\{\mathbf{xp2}, \mathbf{yp2}\}]\}]$$

Полученный контурный график представлен на предыдущем рисунке справа. Напомним, что начальное приближение в области ограничений (если оно не задано) функция `NMaximize` выбирает самостоятельно.

3.7 Приближение функций.

Когда функция задана в дискретном наборе точек и требуется определить ее значение в промежуточных точках, то прибегают к интерполяции или аппроксимации. При этом неизвестная истинная функция заменяется некоторой другой, часто кусочной, имеющей достаточно простые выражения на кусках. Если приближающая функция принимает заданные значения в узлах, то говорят об интерполяции, если нет, то говорят об аппроксимации.

3.7.1 Полиномиальная интерполяция

Функция `InterpolatingPolynomial` создает полином (выражение, степенной многочлен), который проходит через узловые точки. В одномерном случае по `n` точкам строится полином степени `n-1`.

При вызове функции `InterpolatingPolynomial` ей передается список координат узлов и идентификатор переменной полинома, например, *x*, поскольку возвращаться будет выражение, а не функция. При этом интерполяционный полином возвращается в форме Горнера. Например,

$\mathbf{InterpolatingPolynomial}[\{1, 8, 27, 64, 125\}, x]$

$1 + (-1 + x)(7 + (-2 + x)(3 + x))$

Здесь список `{1,8,27,64,125}` представляет ординаты узловых точек; их абсциссами полагаются натуральные числа `{1,2,3,...}`. Результирующий полином всегда можно преобразовать к обычному виду.

$\mathbf{Expand}[\%]$

x^3

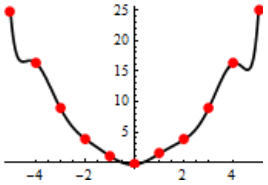
Существует бесчисленное множество полиномов, проходящих через заданные точки. Функция `InterpolatingPolynomial` пытается возвратить полином наименьшей степени.

В формате `InterpolatingPolynomial[{{x1,y1},{x2,y2},...},x]` задаются обе координаты узловых точек.

В следующем коде мы выполняем полиномиальную интерполяцию по множеству точек, приближенно расположенных на параболе $y = x^2$; смещение с параболы задается небольшим случайным значением.

```
SeedRandom[ ];
data = Table[{i, i^2 + RandomReal[{-0.5, 0.5}]}, {i, -5, 5}]
g[x_] = Collect[InterpolatingPolynomial[data, x], x]
Plot[g[x], {x, -5, 5}, PlotStyle → {Black, Thickness[0.01]},
                Epilog-> {Red, PointSize[0.04], Point/@data}]
```

```
-0.29820805+0.33587531 x+1.9489638 x^2 -0.18505705 x^3-
0.42240753 x^4+0.029138328 x^5+0.061215827 x^6-0.0017099349 x^7-
0.0034036856 x^8+0.000032814108 x^9+0.000062829842 x^10
```

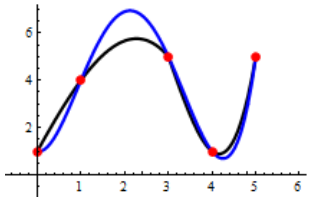


Здесь функция `SeedRandom[ ]` инициализирует генератор случайных чисел текущим временем.

В следующем пункте мы рассмотрим кусочно – полиномиальную интерполяцию. Забегая вперед, выполним сравнение кусочно – полиномиальной интерполяции, выполняемой функцией `Interpolation`, и полиномиальной, выполняемую функцией `InterpolatingPolynomial`.

```
data = {{0, 1}, {1, 4}, {3, 5}, {4, 1}, {5, 5}};
f = Interpolation[data];
g[x_] = Collect[InterpolatingPolynomial[data, x], x]
Show[Plot[{f[x], g[x]}, {x, 0, 5}, PlotStyle → {Black, Blue}],
     ListPlot[data, PlotStyle → {Red, PointSize[0.03]}],
     PlotRange → {{-1, 6}, {-1, 7}}, AxesOrigin → {0, 0}]
```

$$1 - \frac{11x}{30} + \frac{683x^2}{120} - \frac{79x^3}{30} + \frac{37x^4}{120}$$



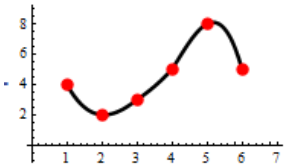
3.7.2 Кусочно – полиномиальная интерполяция

Для представления результатов интерполяции в *Mathematica* используется специальный объект `InterpolatingFunction`. Это «чистая» функция, которую можно использовать как обыкновенную функцию: вычислять значение в точке, дифференцировать и т.д. Единственное ограничение состоит в том, что она создается на определенном диапазоне изменения аргументов. При попытке подстановки аргумента, выходящего за границы этого диапазона, генерируется сообщение.

InterpolatingFunction выполняет кусочно – полиномиальную интерполяцию, порядок гладкости которой можно задать при создании функции. Она обычно создается другими функциями системы, выполняющими приближенные вычисления. В частности при численном решении ДУ с помощью функции NDSolve создается приближенное решение в виде такой функции.

Алгебраическую интерполяцию в системе выполняет функция Interpolation, которая принимает набор точек и создает объект InterpolatingFunction.

```
data = {4, 2, 3, 5, 8, 5};
f = Interpolation[data]
f[2.5]
Show[Plot[f[x], {x, 1, 6}, PlotStyle → {Black, Thickness[0.015]}],
     ListPlot[data, PlotStyle → {Red, PointSize[0.05]}],
     PlotRange → {{0, 7}, {−1, 9}}, AxesOrigin → {0, 0}]
InterpolatingFunction[{{1,6}},"<>"]
2.25
```



Здесь мы использовали формат вызова Interpolation[{y_1,y_2,...}], который полагает, что абсциссы точек равны {1,2,...}, а ординаты – {y_1,y_2,...}.

При выводе функции InterpolatingFunction[{{1,6}},"<>"] показан диапазон интерполяции {1,6}. Именно его мы использовали в качестве диапазона при построении графика функции f[x]. Попробуйте использовать более широкий диапазон Plot[f[x],{x,0,7},... и вы получите предупреждающее сообщение, хотя график будет построен.

Если нужно вычислить значение интерполяционной функции в точке, то можно использовать следующий формат Interpolation[data,$x_{точки}$].

```
Interpolation[data, 2.5]
2.25
```

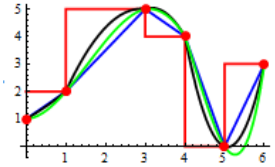
В формате вызова Interpolation[{{x_1,y_1},{x_2,y_2},...}] задаются обе координаты точек.

```
pnt = {{0, 1}, {1, 2}, {4, 4}, {3, 5}, {5, 0}, {6, 3}};
f = Interpolation[pnt]
f[2.5]
Plot[f[x], {x, 0, 6}, PlotStyle → {Black, Thickness[0.015]},
          Epilog−> {Red, PointSize[0.04], Point/@pnt}]
InterpolatingFunction[{{0,6}},"<>"]
4.59375
```

Обратите внимание, что абсциссы точек не обязаны следовать в возрастающем порядке.

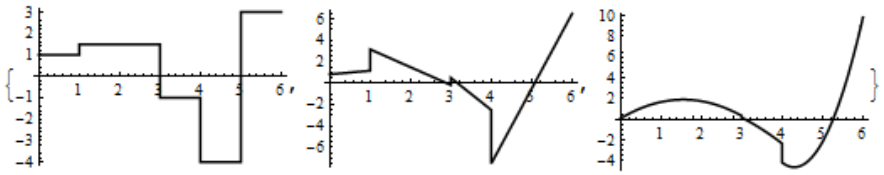
На участках между исходными точками интерполяционная функция представляется полиномами, степень которых можно задавать с помощью опции `InterpolationOrder`. В следующем коде на одном наборе точек мы создаем четыре функции с порядком интерполяции 0, 1, 2, 3 и строим их графики.

```
pnt = {{0, 1}, {1, 2}, {4, 4}, {3, 5}, {5, 0}, {6, 3}};
Table[ToExpression["f" <> ToString[i]][x_] =
      Interpolation[pnt, InterpolationOrder → i][x], {i, 0, 3}];
Plot[{f0[x], f1[x], f2[x], f3[x]}, {x, 0, 6},
     PlotStyle → {{Red, Thickness[0.01]}, {Blue, Thickness[0.01]},
                  {Black, Thickness[0.01]}, {Green, Thickness[0.01]}},
     Epilog−> {Red, PointSize[0.04], Point/@pnt}]
```



Нулевой порядок создает кусочно – постоянную интерполяционную функцию, первый – непрерывную кусочно – линейную и т.д. Однако это не сплайн – интерполяция. Это видно из графиков производных.

```
{Plot[Evaluate[D[f1[x], x]], {x, 0, 6}, PlotStyle → {Black, Thickness[0.01]}],
 Plot[Evaluate[D[f2[x], x]], {x, 0, 6}, PlotStyle → {Black, Thickness[0.01]}],
 Plot[Evaluate[D[f3[x], x]], {x, 0, 6}, PlotStyle → {Black, Thickness[0.01]}]}
```



Обычно при сплайн – интерполяции `n` – го порядка в точках стыка сегментов кусочного полинома выполняется непрерывность производных до порядка `n-1` включительно. Из графиков видно, что уже первая производная интерполяционных функций имеет разрывы в точках стыка. Ниже мы узнаем, что по умолчанию выполняется интерполяция Эрмита.

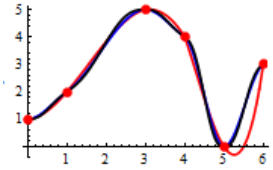
В связи со сказанным, имеется формат вызова функции `Interpolation`, в котором можно задавать значения производных в узловых точках. Этот формат имеет вид

```
Interpolation[{{{x1}, y1, y1', ...}, {{x2}, y2, y2', ...}, ...}],
```

где x_i – абсцисса i-ой точки, y_i – ордината этой точки, y_i' – значение первой производной и т.д.

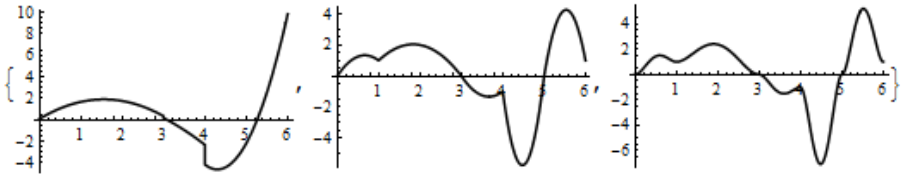
В следующем примере мы строим три интерполяционные функции 3 – го порядка на одном и том же наборе точек $\{x_i, y_i\}$. Для функции f3 мы не задаем значение производных, для функции g3 – задаем значения первых производных, для функции h3 – задаем значения первых и вторых производных в узловых точках.

```
pnt0 = {{0, 1}, {1, 2}, {4, 4}, {3, 5}, {5, 0}, {6, 3}};
pnt1 = {{{0}, 1, 0}, {{1}, 2, 1}, {{4}, 4, -1}, {{3}, 5, 0}, {{5}, 0, 0}, {{6}, 3, 1}};
pnt2 = {{{0}, 1, 0, 0}, {{1}, 2, 1, 0}, {{4}, 4, -1, 0}, {{3}, 5, 0, 0},
                                                {{5}, 0, 0, 0}, {{6}, 3, 1, 0}};
f3 = Interpolation[pnt0, InterpolationOrder → 3]
g3 = Interpolation[pnt1, InterpolationOrder → 3]
h3 = Interpolation[pnt2, InterpolationOrder → 3]
Plot[{f3[x], g3[x], h3[x]}, {x, 0, 6}, PlotStyle → {{Red, Thickness[0.01]},
                {Blue, Thickness[0.01]}, {Black, Thickness[0.01]}},
      Epilog-> {Red, PointSize[0.04], Point/@pnt}]
```



Как видим, кривые не очень отличаются. Однако, если построить графики первых производных этих функций, то различие существенное.

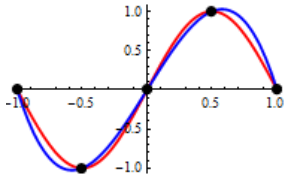
```
{Plot[Evaluate[D[f3[x], x]], {x, 0, 6}, PlotStyle → {Black, Thickness[0.01]}],
 Plot[Evaluate[D[g3[x], x]], {x, 0, 6}, PlotStyle → {Black, Thickness[0.01]}],
 Plot[Evaluate[D[h3[x], x]], {x, 0, 6}, PlotStyle → {Black, Thickness[0.01]}]}
```



Как видно, производная функции f3 имеет разрыв, функция g3′ непрерывна, но имеет резкие изломы, функция h3' уже гладкая.

Проверим качество интерполяции следующим примером, задав узлы на синусоиде.

```
ts = Chop[Table[{x, Sin[πx]}, {x, -1, 1, 0.5}]]
f = Interpolation[ts];
Plot[{Sin[πx], f[x]}, {x, -1, 1},
      PlotStyle → {{Red, Thickness[0.01]}, {Blue, Thickness[0.01]}},
      Epilog-> {Black, PointSize[0.04], Point/@ts}]
{{-1.,0}, {-0.5, -1.}, {0,0}, {0.5,1.}, {1.,0}}
```



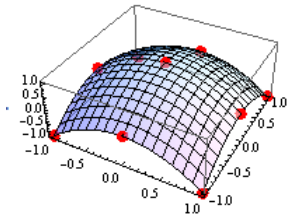
Если в этом примере шаг точек уменьшить, то графики будут сливаться.

Можно приближать многомерные данные. Например в двумерном случае формат вызова может быть следующим

`Interpolation[{{{x_i, y_i}, z_i}, {{x_2, y_2}, z_2}}, ...}],`

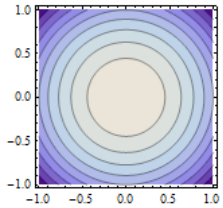
где $\{x_i, y_i\}$ – координаты точки на плоскости, а z_i – значение функции в этой точке.

```
data = Flatten[Table[{{x, y}, 1 - x^2 - y^2}, {x, -1, 1, 1}, {y, -1, 1, 1}], 1];
surf = Interpolation[data, InterpolationOrder → 2]
Show[Plot3D[surf[x, y], {x, -1, 1}, {y, -1, 1}, PlotStyle → Opacity[.5]],
     Graphics3D[{Red, PointSize[0.05],
                 Map[Point, Partition[Flatten[data], 3]]}]]
```



В многомерном случае, кроме значения функции в узлах можно задавать обе частные производные (т.е. в узле задавать градиент).

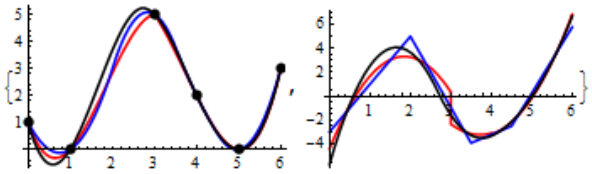
```
z[x_, y_] = 1 - x^2 - y^2;
data = Flatten[Table[Evaluate[{{x, y}, z[x, y], D[z[x, y], {{x, y}}]}],
{x, -1, 1, 0.5}, {y, -1, 1, 0.5}], 1]
 {{{-1., -1.}, -1., {2., 2.}}, {{-1., -0.5}, -0.25, {2., 1.}}, ...
f = Interpolation[data]
ContourPlot[f[x, y], {x, -1, 1}, {y, -1, 1}]
```



У функции `Interpolation` есть опция `Method`, которая задает метод полиномиальной интерполяции. Это может быть сплайн – интерполяция или интерполяция Эрмита. В следующем примере мы строим интерполяционные функции, используя оба метода, причем строим сплайны 2 – го и 3 – го порядков.

```
pnt0 = {{0, 1}, {1, 0}, {4, 2}, {3, 5}, {5, 0}, {6, 3}};
FH = Interpolation[pnt0, Method → "Hermite"];
FS2 = Interpolation[pnt0, Method → "Spline", InterpolationOrder → 2];
FS3 = Interpolation[pnt0, Method → "Spline", InterpolationOrder → 3];
{Plot[{FH[x], FS2[x], FS3[x]}, {x, 0, 6}, PlotStyle → {Red, Blue, Black},
```

```
    Epilog → {Black, PointSize[0.04], Point/@pnt0}],
Plot[Evaluate[{D[FH[x], x], D[FS2[x], x], D[FS3[x], x]}], {x, 0, 6},
    PlotStyle → {Red, Blue, Black}]}
```



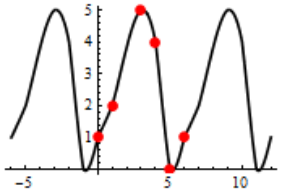
Здесь на графике слева показаны интерполяционные кривые, а справа – их производные. Из правого графика видно, что интерполяционная функция Эрмита **FH** имеет разрыв производной. Производная сплайна 2 – го порядка **FS2** непрерывна, но имеет изломы. Гладкость производной сплайна 3 – го порядка **FS3** говорит о том, что у него непрерывна 2 – я производная.

Использование сплайн интерполяции невозможно, когда в узлах задаются значения производных. Например следующий код выводит сообщение о том, что сплайн – интерполяция невозможна. После этого автоматически выполняется интерполяция Эрмита.

```
pnt1 = {{{0}, 1, 0}, {{1}, 0, 1}, {{4}, 2, -1}, {{3}, 5, 0}, {{5}, 0, 0}, {{6}, 3, 1}};
FS = Interpolation[pnt1, Method → Spline];
FS[2.5]
Interpolation::mspl: The Spline method could not be used because
the data could not be coerced to machine real numbers
4.3125
```

Имеется еще опция `PeriodicInterpolation`. Если она принимает значение `True`, то строится периодическая интерполирующая функция.

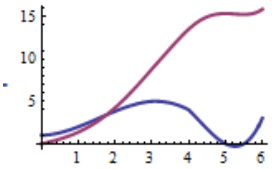
```
pnt0 = {{0, 1}, {1, 2}, {4, 4}, {3, 5}, {5, 0}, {6, 1}};
f = Interpolation[pnt0, PeriodicInterpolation → True];
Plot[f[x], {x, -6, 12}, PlotStyle → {Black, Thickness[0.01]},
                    Epilog-> {Red, PointSize[0.04], Point/@pnt0}]
```



При использовании этой опции следует помнить, что значения в первом и последнем узле должны задаваться одинаковыми.

Можно интегрировать интерполяционную функцию. В следующем коде мы строим графики интерполяционной кривой и ее первообразной.

```
pnt0 = {{0, 1}, {1, 2}, {4, 4}, {3, 5}, {5, 0}, {6, 3}};
f = Interpolation[pnt0];
F[x_] = Integrate[f[x], x]
Plot[{f[x], F[x]}, {x, 0, 6}, PlotStyle → Thickness[0.01]]
```



Сделаем еще несколько замечаний о функции `Interpolation`.

Когда порядок интерполяции задан по умолчанию, нужно передавать не менее 4 – х узлов.

$$\mathbf{Interpolation}[\{\{0,1\},\{1,2\},\{4,4\}\}];$$

```
Interpolation::inhr: Requested order is too high; order has been
reduced to {2}.
```

Интерполяционная функция всегда создается непрерывной, но необязательно дифференцируемой.

Значения функции можно задавать комплексными, но координаты точек должны быть вещественными.

Элементы структуры объекта `InterpolatingFunction[{{1,6}},"<>"]` можно проанализировать с помощью индексации. Эти элементы обычно не отображаются в выводе, но они всегда присутствуют.

$$\mathbf{pnt0} = \{\{0,1\},\{1,2\},\{3,5\},\{4,4\},\{5,0\},\{6,1\}\};$$

$$f = \mathbf{Interpolation}[\mathbf{pnt0}, \mathbf{InterpolationOrder} \to 2];$$

$$\{f[[0]], f[[1]], f[[2]], f[[3]], f[[4]], f[[5]]\}$$

```
InterpolatingFunction[{{0,6}},<>]
{InterpolatingFunction,{{0,6}},
 {4,3,0,{6},{3},0,0,0,0,Automatic},
 {{0,1,3,4,5,6}},{{1},{2},{5},{4},{0},{1}},{Automatic}}
```

Элемент `f[[1]]` содержит диапазон `{{0,6}}`, на котором построена интерполяционная функция. Элемент `f[[3]]` содержит абсциссы узлов, а элемент `f[[4]]` содержит их ординаты (при интерполяции по Эрмиту). Меняя данные и значения опций, можно сделать вывод, что элемент `f[[2,4]]` хранит количество узлов, элемент `f[[2,5]]` на единицу больше порядка интерполяции `InterpolationOrder`.

3.7.3 Аппроксимация

В *Mathematica* имеются функции, которые решают близкие к интерполяции по смыслу задачи.

Функция `Fit` выполняет приближение методом наименьших квадратов набора данных с помощью линейной комбинации заданного набора функций.

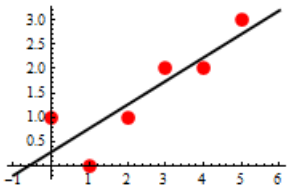
Например, пусть заданы точки и требуется приблизить их линейной функцией. Это можно сделать следующим образом.

$$\mathbf{data} = \{\{0,1\},\{1,0\},\{2,1\},\{3,2\},\{4,2\},\{5,3\}\};$$

$$\mathbf{line} = \mathbf{line} = \mathbf{Fit}[\mathbf{data}, \{1, x\}, x]$$

$$0.28571429 + 0.48571429\, x$$

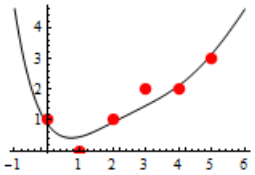
$$\mathbf{Show}[\mathbf{Plot}[\mathbf{line}, \{x, -1, 6\}, \mathbf{PlotStyle} \to \{\mathbf{Black}, \mathbf{Thickness}[0.01]\}],$$
$$\mathbf{Graphics}[\{\mathbf{Red}, \mathbf{PointSize}[0.05], \mathbf{Point}[\mathbf{data}]\}]]$$



Здесь для приближения мы использовали две функции: 1 (единица) и x. Для приближения тех же данных можно использовать другие функции или больший набор «пробных» функций. В следующем примере мы используем четыре функции: 1 (единица), x, $\sin x$ и e^{-x}.

```
line = Fit[data, {1, x, Sin[x], Exp[-x]}, x]
Show[Plot[line, {x, -1, 6}, PlotStyle → {Black, Thickness[0.01]}],
      Graphics[{Red, PointSize[0.05], Point[data]}],
      AxesOrigin → {0, 0}, PlotRange → All]
```

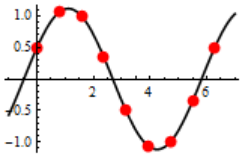
```
-2.2151282+3.0802025 e^-x+1.1566815 x+0.45308887 Sin[x]
```



В следующем примере мы выбираем дискретный набор точек на кривой $\sin x + \frac{1}{2}\cos x$ и аппроксимируем его функциями $\sin x$ и $\cos x$.

```
data = Table[{x, Sin[x] + 1/2 Cos[x]}, {x, 0, 2π, π/4}]
line = Chop[Fit[data, {1, x, Sin[x], Cos[x]}, x]]
Show[Plot[line, {x, -1, 7}, PlotStyle → {Black, Thickness[0.01]}],
      Graphics[{Red, PointSize[0.05], Point[data]}],
      AxesOrigin → {0, 0}, PlotRange → All]
```

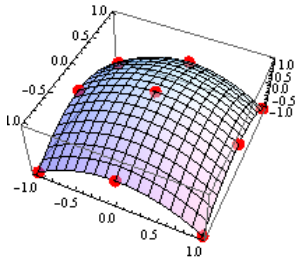
```
0.5 Cos[x]+1.0 Sin[x]
```



После отбрасывания погрешностей вычислений функцией `Chop` мы получили исходное выражение. Как видим, приближение очень хорошее.

Можно приближать множества точек в пространстве с помощью функций 2 – х переменных.

```
data = Flatten[Table[{x, y, 1 - x^2 - y^2}, {x, -1, 1, 1}, {y, -1, 1, 1}], 1]
surf = Chop[Fit[data, {1, x, y, x^2, y^2, xy}, {x, y}]]
{{-1, -1, -1}, {-1,0,0}, {-1,1, -1}, {0, -1,0},
                    {0,0,1}, {0,1,0}, {1, -1, -1}, {1,0,0}, {1,1, -1}}
 1.-1.0 x^2-1.0 y^2
Show[Plot3D[surf, {x, -1, 1}, {y, -1, 1}, PlotStyle → Opacity[.5]],
      Graphics3D[{Red, PointSize[0.05], Map[Point, data]}]]
```



С другими функциями системы близкими к рассмотренным по смыслу, вы можете познакомиться по справочной системе.

3.8 Решение прикладных задач.

Приложения математического анализа неисчислимы. Здесь мы выбрали несколько тем, которые демонстрируют возможности описанных в данной главе функций системы.

3.8.1 Физические приложения кратных интегралов.

Центр тяжести тела. Пусть $\rho = \rho(x, y)$ представляет плотность материала двумерного тела (масса единицы площади в точке *x*,*y*). Тогда масса тела, занимающего область *D* на плоскости, вычисляется по формуле

$$M = \iint_D \rho(x, y)\, d\,x\, d\,y \qquad (1)$$

Положение (x_c, y_c) центра тяжести тела определяется по формулам

$$x_c = \frac{1}{M}\iint_D x\,\rho(x, y)\, d\,x\, d\,y,\ \ y_c = \frac{1}{M}\iint_D y\,\rho(x, y)\, d\,x\, d\,y \qquad (2)$$

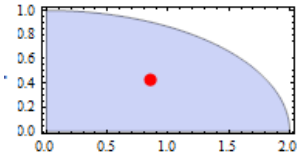
Когда говорят о центре тяжести фигуры, то подразумевают, что плотность $\rho(x, y) = 1$.

Пример. Найти центр тяжести первого квадранта эллипса. Имеем

```
Clear[a, b];
opt = Sequence[{x, 0, a}, {y, 0, b}, Assumptions -> a > 0 && b > 0];
rg = x^2/a^2 + y^2/b^2 <= 1 && x >= 0 && y >= 0;
M = Integrate[Boole[rg], opt];
Mx = Integrate[x Boole[rg], opt];
My = Integrate[y Boole[rg], opt];
{xc, yc} = {Mx, My}/M
```

$$\left\{\frac{4a}{3\pi}, \frac{4b}{3\pi}\right\}$$

```
a = 2; b = 1;
rg = RegionPlot[rg, {x, 0, a}, {y, 0, b}, AspectRatio -> Automatic];
pt = Graphics[{Red, PointSize[0.05], Point[{xc, yc}]}];
Show[rg, pt]
```



Пример. Найти центр тяжести фигуры, ограниченной ветвью циклоиды $x=a(t-\sin t)$, $y=a(1-\cos t)$ и осью x.

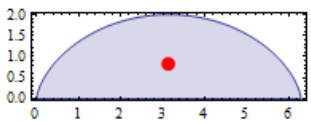
В формуле Грина $\iint\limits_D\left(\frac{\partial Q}{\partial x}-\frac{\partial P}{\partial y}\right)dx\,dy=\oint\limits_L P\,dx+Q\,dy$ положим $Q=x^2/2, P=0$. Тогда она принимает вид $\iint\limits_D x\,dx\,dy=\oint\limits_L\frac{x^2}{2}dy$, где L – контур, составленный из циклоиды и отрезка $[0,2\pi]$ оси Ox, на котором $dy=0$. Поэтому под контуром L можно понимать только дугу циклоиды. Аналогично, полагая $Q=0, P=-y^2/2$, получаем $\iint\limits_D y\,dx\,dy=-\oint\limits_L\frac{y^2}{2}dx$, где L – контур, составленный из циклоиды и отрезка $[0,2\pi]$ оси Ox, на котором $y=0$ и, поэтому под контуром L можно понимать только одну дугу циклоиды. В результате формулы (2) принимают вид $x_c=\frac{1}{2M}\oint\limits_L x^2\,dy$ и $y_c=-\frac{1}{2M}\oint\limits_L y^2\,dx$, где M определяется как и раньше по (1). Заменяя x и y их параметрическим представлением, приходим к необходимым формулам вычисления центра тяжести нашей фигуры.

```
a =.; x = a(t - Sin[t]); y = a(1 - Cos[t]);
M = -Integrate[xD[y, t], {t, 0, 2π}];
Mx = -Integrate[x^2 D[y, t], {t, 0, 2π}]/2;
My = -Integrate[-y^2 D[x, t], {t, 0, 2π}]/2;
{xc, yc} = {Mx, My}/M
```

$$\{a\,\pi,\frac{5a}{6}\}$$

Обратите внимание на минусы, которые необходимо поставить перед выражениями для M, Mx и My, поскольку циклоида обходит область в направлении по часовой стрелки, т.е. в отрицательном направлении.

```
a = 1;
rg = ParametricPlot[{vx, vy}, {t, 0, 2π}, {v, 0, 1}, Mesh → None];
pt = Graphics[{Red, PointSize[0.05], Point[{xc, yc}]}];
Show[rg, pt]
```



□

Положение центра тяжести трехмерного тела определяется по формулам

$$x_c = \frac{1}{M}\iiint_V \rho\, x\, d\,x\, d\,y\, d\,z,\ y_c = \frac{1}{M}\iiint_V \rho\, y\, d\,x\, d\,y\, d\,z,\ z_c = \frac{1}{M}\iiint_V \rho\, z\, d\,x\, d\,y\, d\,z,$$

где $M = \iiint_V \rho\, d\,x\, d\,y\, d\,z$ представляет массу тела.

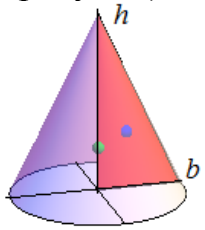
Пример. Найдем центр массы полушара $x^2 + y^2 + z^2 \le 1\ \ (z \ge 0)$ плотности $\rho = 1$. В силу симметрии $\iiint_V x\, d\,x\, d\,y\, d\,z = 0$ и $\iiint_V y\, d\,x\, d\,y\, d\,z = 0$. Далее имеем

```
Mz = Integrate[z Boole[x^2 + y^2 + z^2 <= 1 && z >= 0],
                                {x, -1, 1}, {y, -1, 1}, {z, 0, 1}]
M = Integrate[Boole[x^2 + y^2 + z^2 <= 1 && z >= 0],
                                {x, -1, 1}, {y, -1, 1}, {z, 0, 1}]
zc = Mz/M
```

$$\frac{3}{8}$$

Т.о. центр массы единичного полушара находится в точке $\left(0, 0, \frac{3}{8}\right)$.

Пример. Определить положение центра тяжести плоского треугольника, и конуса, образованного вращением этого треугольника вокруг одного из катетов на оси *Oz*. Вершина треугольника находится в точке (0,0,*h*), а длина катета основания равна *b* (см. следующий рисунок)



Вычисляем положение центра тяжести треугольника

```
Clear[h, b, x, y, z];
brg = Boole[y <= h (1 - x/b) && x >= 0 && y >= 0];
opt = Sequence[{x, 0, b}, {y, 0, h}, Assumptions :> b > 0 && h > 0];
M = Integrate[brg, opt];
Mx = Integrate[x * brg, opt];
My = Integrate[y * brg, opt];
{xc, zc} = {Mx, My}/M
```

$$\left\{\frac{b}{3}, \frac{h}{3}\right\}$$

Вычисляем положение центра тяжести конуса. В силу симметрии его центр тяжести лежит на оси Oz.

```
crg = Boole[0 <= z <= h(1 - Sqrt[x^2 + y^2]/b)];
opt = Sequence[{x, -∞, ∞}, {y, -∞, ∞}, {z, 0, h},
                                Assumptions :> b > 0 && h > 0];
MC = Integrate[crg, opt];
```

```
MCz = Integrate[z * crg, opt];
Zc = MCz/MC
```
$\frac{h}{4}$

Как видим вертикальная координата центра тяжести конуса отличается от вертикальной координаты центра тяжести треугольника.

Нарисуем наши фигуры и положения их центров тяжести (см. предыдущий рисунок)

```
h = 2; b = 1;
ps = Plot3D[h(1 - Sqrt[x^2 + y^2]/b), {x, -b, b}, {y, -b, b},
      RegionFunction → Function[{x, y, z}, x^2 + y^2 ≤ b^2],
      BoxRatios → {2b, 2b, h}, PlotStyle → Opacity[0.5], Mesh → None];
pt = Graphics3D[{Red, Polygon[{{0, 0, 0}, {b, 0, 0}, {0, 0, h}}],
                 Blue, Sphere[{xc, 0, zc}, 0.07],
                 Green, Sphere[{0, 0, Zc}, 0.07]}];
Show[ps, pt, Boxed → False, AxesOrigin → {0, 0, 0}, Ticks → None]
```

□

Механическая работа.

Работа при перемещении тела в силовом поле $\mathbf{F} = (P(x,y,z), Q(x,y,z), R(x,y,z))$ вдоль кривой L выражается криволинейным интегралом

$$A = \oint_L \mathbf{F} \cdot d\,\mathbf{r} = \oint_L P\,dx + Q\,dy + R\,dz$$

При движении в плоскости z=0 третье слагаемое в криволинейном интеграле отсутствует. Если траектория движения L задана параметрически $x = x(t),\ y = y(t)$, то формула принимает вид

$$A = \int_{t_0}^{t_1} \left(P(x(t), y(t))x'(t) + Q(x(t), y(t))y'(t)\right)dt,$$

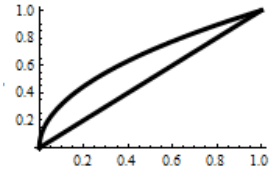
где t_0 и t_1 определяют положение начальной и конечной точек кривой.

Пример. Найти работу поля $\mathbf{F} = (xy, x+y)$ при перемещении тела из начала координат $O(0,0)$ в точку $A(1,1)$ по кривой L_1 (отрезок прямой $y = x$) и по кривой L_2 (отрезок параболы $y^2 = x$).

```
y1[x_] = x;
y2[x_] = Sqrt[x];
Work = Integrate[x * #, {x, 0, 1}] + Integrate[(x + #)D[#, x], {x, 0, 1}]&;
Work[y1[x]]
Work[y2[x]]
```
$\frac{4}{3}$

$\frac{37}{30}$

```
Plot[{y1[x], y2[x]}, {x, 0, 1}, PlotStyle → {{Black, Thickness[0.02]}}]
```



Момент инерции.

Моментом инерции материальной точки относительно оси называется произведение ее массы на квадрат расстояния точки до этой оси.

Для двумерных тел плотностью $\rho(x,y)$ момент инерции относительно осей O*x* и O*y* вычисляется по формулам

$$Ix = \iint\limits_D \rho(x,y)\,y^2\,dx\,dy, \quad Iy = \iint\limits_D \rho(x,y)\,x^2\,dx\,dy$$

При вычислении моментов инерции геометрических фигур полагаем $\rho = 1$.

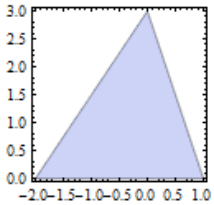
Пример. Найти момент инерции треугольника с основанием b и высотой h относительно его основания.

Основание треугольника примем за ось O*x*, а его высоту – за ось O*y*. Абсциссы точек пересечения боковых сторон с осью O*x* обозначим b_1 и b_2 $(b = b_2 - b_1)$. Имеем

```
Clear[b1, b2, b, h];
Ix0 = Integrate[y^2 Boole[x/b1 + y/h <= 1 && x/b2 + y/h <= 1 && y >= 0],
    {x, -∞, ∞}, {y, -∞, ∞}, Assumptions :→ h > 0 && b1 < 0 && b2 > 0];
Ix = Simplify[Ix0]/.(b2 - b1) → b
```

$$\frac{bh^3}{12}$$

```
b1 = -2; b2 = 1; h = 3;
RegionPlot[x/b1 + y/h <= 1 && x/b2 + y/h <= 1 && y >= 0, {x, b1, b2}, {y, 0, h}]
```



Пример. Найти момент инерции эллипса $\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$ относительно его главных осей.

```
Clear[x, y, a, b];
Ix = Integrate[y^2 Boole[x^2/a^2 + y^2/b^2 <= 1], {x, -∞, ∞}, {y, -∞, ∞},
    Assumptions :→ a > 0 && b > 0]
Iy = Integrate[x^2 Boole[x^2/a^2 + y^2/b^2 <= 1], {x, -∞, ∞}, {y, -∞, ∞},
    Assumptions :→ a > 0 && b > 0]
```

$$\frac{1}{4} a\, b^3\, \pi$$

$$\frac{1}{4}a^3\ b\ \pi$$

3.8.2 Примеры решения задач распространения тепла

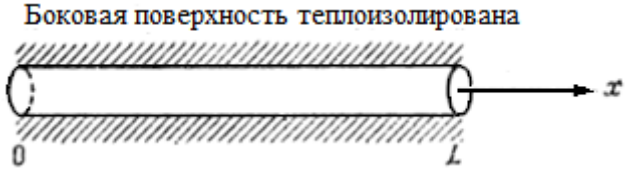
Задача о распространении тепла в тонком однородном неограниченном стержне, боковая поверхность которого теплоизолирована (тепло может распространяться только вдоль оси x), математически формулируется следующим образом [3, 7]: найти ограниченную функцию $u(x,t)$ $(t>0, -\infty<x<\infty)$, удовлетворяющую уравнению теплопроводности

$$\frac{\partial u}{\partial t}=a^2\frac{\partial^2 u}{\partial x^2} \qquad (t>0, -\infty<x<\infty) \tag{1}$$

и начальному условию

$$u(x,0)=\varphi(x) \qquad (-\infty<x<\infty). \tag{2}$$

где $u(x,t)$ температура сечения x стержня в момент времени t (стержень тонкий, это значит, что температура точек любого поперечного сечения x одинакова).



Известно, что общее решение этой задачи представляется интегралом Пуассона следующего вида

$$u(x,t)=\frac{1}{2a\sqrt{\pi t}}\int_{-\infty}^{\infty}\varphi(\xi)e^{-\frac{(\xi-x)^2}{4a^2t}}d\xi \tag{3}$$

Если в этом интеграле сделать замену $\tau=\dfrac{\xi-x}{2a\sqrt{t}}$, то мы получим

$$u(x,t)=\frac{1}{\sqrt{\pi}}\int_{-\infty}^{\infty}\varphi\left(x+2a\sqrt{t}\,\tau\right)e^{-\tau^2}d\tau \tag{4}$$

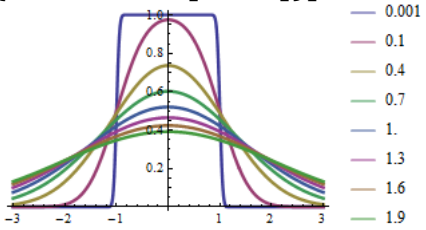
Пример. *Бесконечный стержень. Начальная температура всюду ноль, кроме отрезка* [-1,1], *где она равна единице*, т.е. $u(x,0)=\varphi(x)=\begin{cases}1, -1\le x\le 1\\ 0, x<-1\vee x>1\end{cases}$. В этом примере интеграл (4) вычисляется символьно и выражается через интеграл ошибок.

```
a =.;
φ[x_] = Piecewise[{{0, x ≤ -1}, {1, x < 1}}, 0];
u[x_, t_] = FullSimplify[1/Sqrt[π] Integrate[φ[x + 2a Sqrt[t] τ] Exp[-τ^2],
                                             {τ, -∞, ∞}, Assumptions :→ t > 0 && a > 0]]
```

$$\frac{1}{2}\left(\mathrm{Erf}\left[\frac{1-x}{2a\sqrt{t}}\right]+\mathrm{Erf}\left[\frac{1+x}{2a\sqrt{t}}\right]\right)$$

На следующем рисунке представлены графики температуры $u(x,t)$ в различные моменты времени (коэффициент теплопроводности $a^2=1$).

```
T = Join[{0.001}, Table[t, {t, 0.1, 2, 0.3}]];  a = 1;
Plot[Evaluate[Table[u[x, t], {t, T}]], {x, -3, 3}, PlotLegends → {T},
                PlotStyle → {Thickness[0.01]}]
```



□

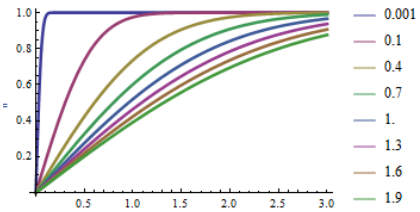
Рассмотрим задачу о распространении тепла в полуограниченном стержне, боковая поверхность которого теплоизолирована, а конец x=0 поддерживается при нулевой температуре, т.е. $u(0,t)=0\ \ (t\ge 0)$. Решение этой задачи может быть получено методом продолжения, когда начальная температура $\varphi(x)$, заданная на положительной полуоси Ox, продолжается на отрицательную полуось нечетным образом. Можно показать [3], что решение в форме интеграла Пуассона (3) с любой нечетной функцией $\varphi(\xi)$ будет представлять решение задачи (1), (2), которое обращается в ноль при x=0. Оно, тем самым, при положительных x будет давать решение поставленной задачи для полубесконечного стержня.

Пример. *Полубесконечный стержень. Начальная температура всюду равна 1, температура на левом конце равна нулю,* т.е. $u(x,0)=1\ \ (x\ge 0)$, $u(0,t)=0\ \ (t\ge 0)$.

Вначале строим нечетную функцию $\varphi(x)=\begin{cases}1, x\ge 0\\ -1, x<0\end{cases}$ и подставляем ее в интеграл (4), который выражается интегралом ошибок. Затем строим графики функции $u(x,t)\ \ (x\ge 0)$ в некоторые фиксированные моменты времени.

```
φ[x_] = Piecewise[{{-1, x ≤ 0}}, 1];  a =.;
u[x_, t_] = FullSimplify[1/Sqrt[π] Integrate[φ[x + 2a Sqrt[t] τ] Exp[-τ^2],
                {τ, -∞, ∞}, Assumptions :→ t > 0 && a > 0]]
T = Join[{0.001}, Table[t, {t, 0.1, 2, 0.3}]]; a = 1;
Plot[Evaluate[Table[u[x, t], {t, T}]], {x, 0, 3}, PlotLegends → {T},
                PlotStyle → {Thickness[0.01]}]
```

$$\text{Erf}\left[\frac{x}{2a\sqrt{t}}\right]$$



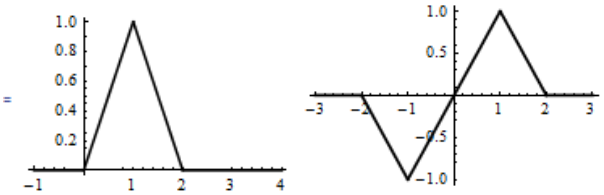
Результирующая последовательность графиков иллюстрирует процесс остывания полубесконечного стержня, левая граница которого поддерживается при нулевой температуре.

□

Пример. *Полубесконечный стержень. Начальная температура везде 0, кроме отрезка у границы, где она имеет треугольную форму. Температура левого конца равна нулю*, т.е. $u(0,t)=0\ \ (t\ge 0)$.

В следующем коде мы строим график функции $\varphi(x)$, которая представляет начальную температуру (следующий рисунок слева) и график функции $\Phi(x)$, которая является ее нечетным продолжением (следующий рисунок справа).

```
φ[x_] = Piecewise[{{0, x ≤ 0}, {x, x < 1}, {2 − x, x < 2}}, 0];
Plot[φ[x], {x, −1, 4}]                    (* график слева *)
Φ[x_] = Piecewise[{{0, x ≤ −2}, {−2 − x, x < −1}, {x, x < 1},
                                  {2 − x, x < 2}}, 0];
Plot[Φ[x], {x, −3, 3}]                    (* график справа *)
```

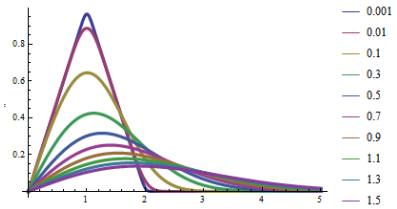


Функцию $\Phi(x)$ используем в интеграле (4) для построения решения задачи. Интеграл вычисляется явно, но из – за длины мы не приводим полный вид его выражения.

```
a =.;
u[x_, t_] = FullSimplify[1/Sqrt[π] Integrate[Φ[x + 2a Sqrt[t] τ] Exp[−τ^2],
                {τ, −∞, ∞}, Assumptions :→ t > 0 && a > 0]]
```

$$\frac{1}{2\sqrt{\pi}}e^{-\frac{(2+x)^2}{4a^2t}}\left(2a\left(-1+e^{\frac{x}{a^2t}}\right)\left(1+e^{\frac{x}{a^2t}}-2e^{\frac{3+2x}{4a^2t}}\right)\sqrt{t}+\cdots\right)$$

```
T = Join[{0.001, 0.01}, Table[t, {t, 0.1, 1.5, 0.2}]];
a = 1;
Plot[Evaluate[Table[u[x, t], {t, T}]], {x, 0, 5},
        PlotStyle → {Thickness[0.01]}, PlotLegends → {T}]
```



□

Пусть начальная температура полубесконечного стержня равна 0 $(u(x,0)=0)$, а на его левом конце задана температура $u(0,t)=\psi(t)$. Решение этой задачи имеет вид ([2], ф.28 стр. 541)

$$u(x,t)=\frac{x}{2a\sqrt{\pi}}\int_0^t\frac{\psi(\tau)}{(t-\tau)^{\frac{3}{2}}}e^{-\frac{x^2}{4a^2(t-\tau)}}d\tau$$

Сделав замену переменной интегрирования $\xi=\frac{x}{2a\sqrt{t-\tau}}$, выражение для функции u приведем к следующему удобному для вычислений виду

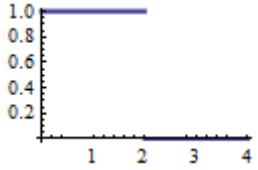
$$u(x,t)=\frac{1}{\sqrt{\pi}}\int_{\frac{x}{2a\sqrt{t}}}^{\infty}\psi(t-\frac{x^2}{4a^2\xi^2})e^{-\xi^2}d\xi \qquad (5)$$

При вычислениях интегралов по этой формуле следует обращать внимание на особенность решения при *t*=0. Но формула «работоспособна» для любых $t>\tau>0$, где τ может быть сколь угодно малыми положительным числом.

Пример. *Полубесконечный стержень. Начальная температура всюду 0, температура на левом конце равна* $u(0,t)=\psi(t)=\begin{cases}1,\ 0\le t\le 2\\ 0,\ t>2\end{cases}$.

Вначале создаем функцию $\psi(t)$, представляющую граничный режим.

```
ψ[t_] = Piecewise[{{0, t < 0}, {1, t ≤ 2}}, 0]; a =.;
Plot[ψ[t], {t, 0, 4}]
```



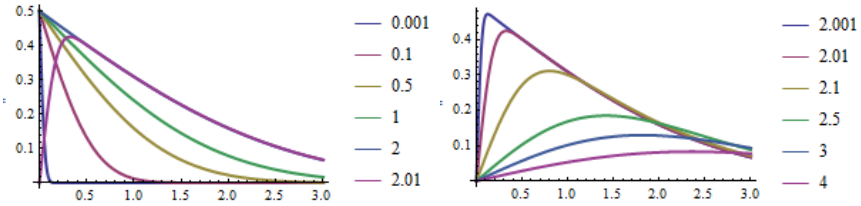
Теперь строим решение, используя интеграл (5).

```
u[x_, t_] = FullSimplify[1/Sqrt[π] Integrate[ψ[t - x^2/(4a^2 ξ^2)] Exp[-ξ^2], {ξ, x/(2a Sqrt[t]), ∞},
                         Assumptions :→ t > 0 && a > 0 && x > 0]]
```

$$\begin{cases}\frac{1}{2}\left(\text{Erf}[\frac{x}{2a\sqrt{-2+t}}]-\text{Erf}[\frac{x}{2a\sqrt{t}}]\right) & a>0\ \&\&\ t>2\ \&\&\ x>0\\ \frac{1}{2}\text{Erfc}[\frac{x}{2a\sqrt{t}}] & \text{True}\end{cases}$$

Решение получено в форме кусочной функции. Теперь построим графики решения для моментов времени $t\le 2$ и для моментов $t>2$.

```
T = {0.001, 0.1, 0.5, 1, 2, 2.01}; a = 1;
Plot[Evaluate[Table[u[x, t], {t, T}]], {x, 0, 3}, PlotLegends → {T},
     PlotStyle → {Thickness[0.01]}]   (* левый рисунок *)
T = {2.001, 2.01, 2.1, 2.5, 3, 4};
Plot[Evaluate[Table[u[x, t], {t, T}]], {x, 0, 3}, PlotLegends → {T},
     PlotStyle → {Thickness[0.01]}]   (* правый рисунок *)
```



Скачкообразное изменение граничной температуры в момент времени $t=2$ меняет качественную картину распределения температуры у торца стержня после этого момента.

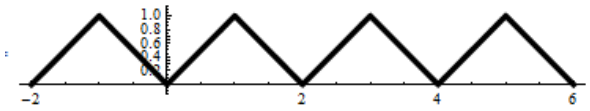
Пример. *Полубесконечный стержень. Начальная температура всюду 0, температура на левом конце периодически меняется по треугольному закону.*

Определим периодическую пилообразную непрерывную кусочно-линейную функцию

$$stx(x,w)=\left|x-w\left[\frac{x}{w}+\frac{1}{2}\right]\right| \quad \text{или} \quad stc(x,w)=\frac{w}{2\pi}\arccos\left(\cos\frac{2\pi x}{w}\right), \qquad (6)$$

где w является периодом этой функции, а квадратные скобки [x] обозначают функцию взятия наибольшего целого, не превосходящего x,

```
stc[x_, w_] = w/(2 π) ArcCos[Cos[(2 π x)/w]];
Plot[stc[x, 2], {x, -2, 6}, AspectRatio → Automatic]
```



Используем ее в качестве граничного значения $u(0,t)=\psi(t)=stc(t,2)$. Поскольку аналитически интеграл (5) не вычисляется, будем его считать численно.

```
a = 1; ψ[t_] := stc[t, 2];
u[x_, t_] := 2/Sqrt[π] NIntegrate[ψ[t - x^2/(4 a^2 ξ^2)] Exp[-ξ^2], {ξ, x/(2 a Sqrt[t]), ∞},
                AccuracyGoal → 4, PrecisionGoal → 4];
```

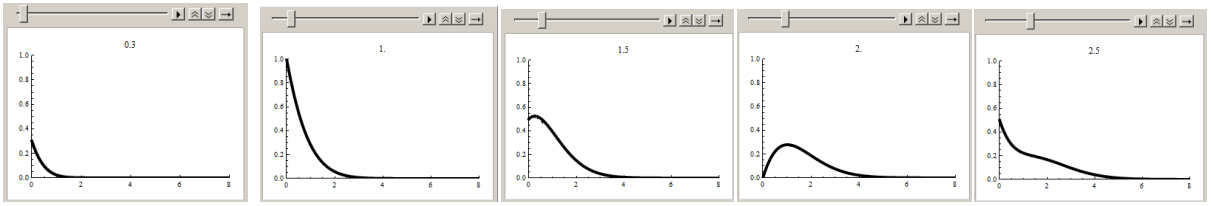
Для построения анимации используем функцию `ListAnimate`. Вначале создаем список графиков **lt**, которые будут представлять кадры анимации. Имейте ввиду, что создание списка **lt** займет некоторое время.

```
lt = Table[Plot[u[x, t], {x, 0, 8}, PlotStyle → {Black, Thickness[0.01]},
             PlotRange → {{0, 8}, {0, 1}}], {t, 0.1, 8, 0.1}];
```

Теперь строим анимацию.

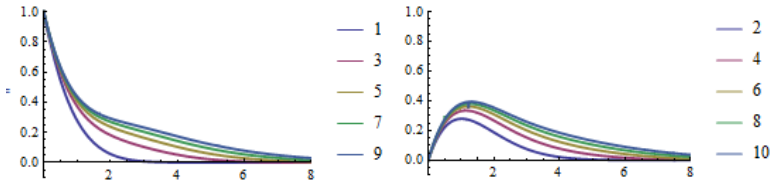
```
ListAnimate[lt, AnimationRunning → False]
```

На следующем рисунке представлено несколько кадров анимации – графиков распределения температуры в полубесконечном стержне в моменты времени $t=0.3, 1.0, 1.5, 2.0, 2.5$



Графики распределения температуры внутри стержня через интервалы времени $\Delta t = 2$ имеют похожую форму. На следующем рисунке слева приведены графики температуры в моменты t = 1,3,5,7,9 и на рисунке справа – в моменты t = 2,4,6,8,10.

```
Plot[{u[x, 1], u[x, 3], u[x, 5], u[x, 7], u[x, 9]}, {x, 0, 8},
    PlotRange → {{0, 8}, {-0.1, 1}}, PlotLegends → {1, 3, 5, 7, 9}]
Plot[{u[x, 2], u[x, 4], u[x, 6], u[x, 8], u[x, 10]}, {x, 0, 8},
    PlotRange → {{0, 8}, {-0.1, 1}}, PlotLegends → {2, 4, 6, 8, 10}]
```



Графики в более поздние моменты времени имеют более высокий профиль.

□

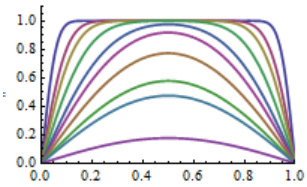
Решение уравнения теплопроводности для стержня конечной длины l, когда начальная температура задана функцией φ(ξ), а концы стержня поддерживаются при нулевой температуре, можно находить по формуле [5]

$$u(x,t) = \frac{1}{2l}\int_0^l \varphi(\xi)\left(\Theta_3\left(\pi\frac{\xi - x}{2l}, e^{-\frac{\pi^2 a^2 t}{l^2}}\right) - \Theta_3\left(\pi\frac{\xi + x}{2l}, e^{-\frac{\pi^2 a^2 t}{l^2}}\right)\right) d\xi, \qquad (7)$$

где $\Theta_3(z,q)$ является тета функция 3-го порядка ([4] стр.334). В пакете *Mathematica* она реализуется функцией `EllipticTheta[3,z,q]`.

Пример. *Конечный стержень длины l. Температура концов ноль. Начальная температура единица.* Вычисления интеграла по формуле (7) реализуем численно.

```
l = 1; a = 1; φ[x_] := 1;
θ3m[τ_, x_, t_] := EllipticTheta[3, π (τ - x)/(2 l), ℯ^(-π^2 a^2 t/l^2)]
θ3p[τ_, x_, t_] := EllipticTheta[3, π (τ + x)/(2 l), ℯ^(-π^2 a^2 t/l^2)]
u[x_, t_] := 1/(2 l) NIntegrate[φ[τ] (θ3m[τ, x, t] - θ3p[τ, x, t]), {τ, 0, l},
                AccuracyGoal → 6, PrecisionGoal → 6]
Plot[{u[x, 0.001], u[x, 0.003], u[x, 0.005], u[x, 0.01], u[x, 0.02],
    u[x, 0.03], u[x, 0.05], u[x, 0.08], u[x, 0.1], u[x, 0.2]}, {x, 0, l},
    PlotRange → {{0, 1}, {0, 1.1}}, PlotStyle → {Thickness[0.01]}]
```

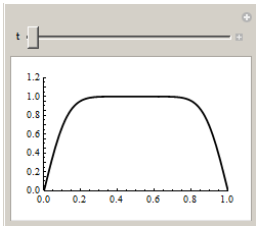


Для решения уравнения теплопроводности на конечном отрезке можно использовать классический метод Фурье [3]. Для рассмотренной нами задачи, когда начальная температура постоянна $\varphi(x)=u_0$, а на обеих границах поддерживается нулевая температура, решение Фурье имеет вид ([6], стр. 54)

$$u=\frac{4u_0}{\pi}\sum_{n=0}^{\infty}\frac{1}{(2n+1)}\sin\left(\frac{(2n+1)\pi x}{l}\right)\exp\left(-\frac{a(2n+1)^2\pi^2 t}{l^2}\right)$$

Построим манипулятор с графиками температуры в стержне в различные моменты времени.

```
u0 = 1; L = 1; a = 1;
u[t_, x_] := 4 u0/π *
             Sum[1/(2n + 1) Sin[(2n + 1) π x/L] Exp[-a(2n + 1)^2 π^2 t/L^2], {n, 0, 30}]
Manipulate[
  Plot[u[t, x], {x, 0, L}, PlotRange → {{0, L}, {0, 1.1}}],
  {t, 0.0001, 0.5}]
```



Обратите внимание, что для ускорения вычислений мы использовали частичную сумму ряда, представляющего решение (суммировали до n=30).

Пример. *Конечный стержень длины l. Начальная температура равна нулю. На границах поддерживаются постоянные температуры* u_1 *и* u_2.

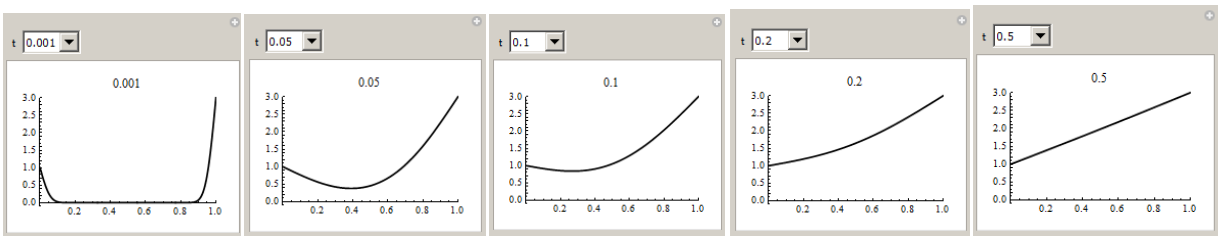
Для поставленной задачи решение Фурье имеет вид ([7], стр. 98, 101, 106)

$$u=u_1+(u_2-u_1)\frac{x}{l}+\frac{2}{\pi}\sum_{n=1}^{\infty}\frac{(-1)^n u_2-u_1}{n}\sin\left(\frac{n\pi x}{l}\right)\exp\left(-\frac{an^2\pi^2 t}{l^2}\right)$$

Представим решение в различные моменты времени, используя манипулятор.

```
L = 1; a = 1; w1 = 1; w2 = 3;
T = Sequence[Join[{0.001}, Table[0.01 * i, {i, 50}]]];
DynamicModule[{u},
  u[t_, x_] := w1 + (w2 - w1) x/L +
    2/π Sum[((-1)^n w2 - w1)/n Sin[n π x/L] Exp[-a n^2 π^2 t/L^2], {n, 1, 50}];
  Manipulate[Plot[u[t, x], {x, 0, L},
    PlotRange -> {{0, L}, {-0.1, w2}}, PlotLabel → t], {t, T}]]
```

На следующем рисунке представлен манипулятор с графиками решения в моменты времени $t = 0.001, 0.05, 0.1, 0.2, 0.5$.

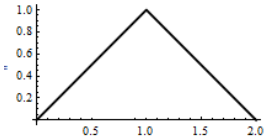


Для ускорения вычислений мы использовали частичную сумму ряда с максимальным значением n=50.

В моменты времени $t \geq 0.4$ графики распределения температуры практически не изменяются и совпадают с последним графиком. Это значит, что в стержне установилась температура $u(x,t) = u_1 + \dfrac{u_2 - u_1}{l} x$ и начиная с момента t=0.4 влияние начального условия пренебрежимо мало.

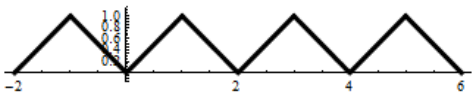
Пример. *Конечный стержень длины l. Температура концов ноль. Начальная температура имеет форму треугольника* $\varphi(x) = 1 - |x - 1|$.

```
φ0[x_] = 1 - Abs[x - 1];
Plot[φ0[x], {x, 0, 2}, AspectRatio → Automatic]
```

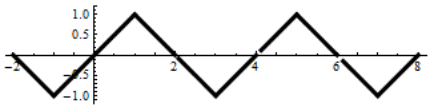


В этом примере мы используем решение для бесконечного стержня (4), используя метод продолжения. Нам надо построить нечетное периодическое продолжение начальной функции $\varphi(x)$ с отрезка [0,2] на всю вещественную ось. Для этого используем функцию $stc(x,w)$, определяемую формулой (6). Если L длина отрезка, то соответствующее продолжение будет иметь вид $\varphi(x) = (-1)^{[x/L]} stc(x,L)$.

```
stc[x_, w_] = Abs[x - w Floor[x/w + 1/2]];
Plot[stc[x, 2], {x, -2, 6}, AspectRatio → Automatic]
```



```
φ[x_] = (-1)^Floor[x/2] stc[x, 2];
Plot[φ[x], {x, -2, 8}, AspectRatio → Automatic]
```



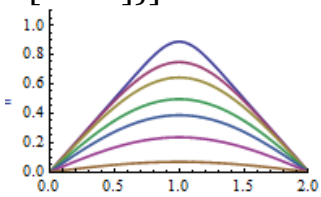
Подставив полученную функцию в (4), мы получим решение для конечного отрезка. Строим решение

```
a = 1;
u[x_, t_] := 1/Sqrt[π] NIntegrate[φ[x + 2 a Sqrt[t] τ] Exp[-τ^2], {τ, -∞, ∞},
```

```
    AccuracyGoal → 4, PrecisionGoal → 4,
    Method → "ClenshawCurtisRule"]
```

и его графики в разные моменты времени

```
Plot[{u[x, 0.01], u[x, 0.05], u[x, 0.1], u[x, 0.2], u[x, 0.3],
    u[x, 0.5], u[x, 1]}, {x, 0, 2}, PlotRange → {{0, 2}, {0, 1.1}},
    PlotStyle → {Thickness[0.01]}]
```

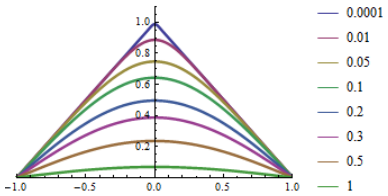


В работе [7] стр. 103 приводится решение в рядах для области $-l \le x \le l$ с начальной температурой $u_0(l-|x|/l)$ и нулевой температурой концов

$$u(x,t)=\frac{8u_0}{\pi^2}\sum_{n=0}^{\infty}\frac{1}{(2n+1)^2}\cos\frac{(2n+1)\pi x}{2l}\exp\left(-\frac{a(2n+1)^2\pi^2 t}{4l^2}\right)$$

Используем его для сравнения с решением, полученным выше.

```
u0 = 1; L = 1; a = 1;
u[x_, t_] :=
    (8 u0)/π^2 Sum[1/(2n + 1)^2 Cos[((2n + 1) π x)/(2L)] Exp[-(a(2n + 1)^2 π^2 t)/(4L^2)], {n, 0, 30}]
T = {0.0001, 0.01, 0.05, 0.1, 0.2, 0.3, 0.5, 1};
Plot[Evaluate[Table[u[x, t], {t, T}]], {x, -1, 1},
    PlotRange → {{-1, 1}, {0, 1.1}}, PlotStyle → {Thickness[0.01]},
    PlotLegends → T]
```



Для ускорения вычислений мы использовали частичную сумму ряда, представляющего решение (суммировали до n=30).

Когда решение в виде ряда известно, то вычисления происходят значительно быстрее.

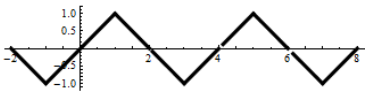
Пример. *Конечный стержень длины L. Температура левого конца ноль. Правый конец теплоизолирован. Начальная температура линейная функция.* Условие теплоизолированности правого конца математически формулируются как $\frac{\partial u(L,t)}{\partial x}=0$. Линейность начальной функции означает, что $u(x,0)=k\,x$.

Продолжим начальную функцию четно относительно точки x=L на отрезок [L,2L], затем полученную функцию с отрезка [0,2L] продолжим нечетно на отрезок [-2L,0], и затем с отрезка [-2L,2L] продолжим периодически на всю ось. Описанное построение в п. 2.4.5 (пример 5.5) мы назвали четно – нечетным

периодическим продолжением функции. Если полученную функцию $\Phi(x)$ подставить в (4), то получим решение нашей краевой задачи.

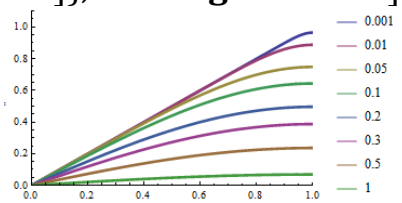
Пусть k=1. Четно – нечетное продолжение линейной функции $y=x$ с отрезка $[0,L]$ на всю ось имеет вид $\varphi(x)=(-1)^{[x/L]}stc(x,L)$, совпадающий с функцией $\varphi(x)$ из предыдущего примера.

```
φ[x_] = (-1)^Floor[x/2]stc[x, 2];
Plot[φ[x], {x, -2, 8}, AspectRatio → Automatic]
```



Эта функция имеет тот же вид, что и в предыдущем примере, только решение теперь надо рассматривать на отрезке [0,1], а не [0,2]. Имеем

```
a = 1;
u[x_, t_] := 1/Sqrt[π] NIntegrate[φ[x + 2a Sqrt[t] τ] Exp[-τ^2], {τ, -∞, ∞},
AccuracyGoal → 4, PrecisionGoal → 4, Method → "ClenshawCurtisRule"]
T = {0.001, 0.01, 0.05, 0.1, 0.2, 0.3, 0.5, 1};
Plot[{u[x, 0.001], u[x, 0.01], u[x, 0.05], u[x, 0.1], u[x, 0.2], u[x, 0.3],
u[x, 0.5], u[x, 1]}, {x, 0, 1}, PlotRange → {{0, 1}, {0, 1.1}},
PlotStyle → {Thickness[0.01]}, PlotLegends → T]
```



Касательная к графику решения на правой границе в любой момент времени горизонтальна, т.е. $u'_x(L,t)=0$. Это означает, что тепловой поток равен нулю.

Пример. *Конечный стержень длины l. Начальная температура ноль. Температура левого конца равна* 1, *правого – ноль.* В соответствии с формулой (7) функция вида

$$\tilde{u}(x,t)=\frac{1}{2l}\int_0^l\left(1-\frac{\xi}{l}\right)\left(\Theta_3\left(\pi\frac{\xi-x}{2l},e^{-\frac{\pi^2a^2t}{l^2}}\right)-\Theta_3\left(\pi\frac{\xi+x}{2l},e^{-\frac{\pi^2a^2t}{l^2}}\right)\right)d\xi$$

представляет решение уравнения теплопроводности с нулевыми граничными условиями и начальной температурой $u(x,0)=1-x/l$. Тогда функция $u(x,t)=1-\frac{x}{l}-\tilde{u}(x,t)$ будет представлять решение нашей задачи, поскольку она удовлетворяет уравнению теплопроводности, на левом конце принимает значение 1, на правом $x=l$ – принимает значение 0, и значение функции $u(x,t)$ при t=0 для всех $0<x<l$ будет 0. Т.о. общее решение нашей задачи будет иметь вид

$$u(x,t)=1-\frac{x}{l}-\frac{1}{2l}\int_0^l(1-\frac{\xi}{l})\left(\Theta_3(\pi\frac{\xi-x}{2l},e^{-\frac{\pi^2a^2t}{l^2}})-\Theta_3(\pi\frac{\xi+x}{2l},e^{-\frac{\pi^2a^2t}{l^2}})\right)d\xi$$

Построим графики распределения температуры в стержне для этой задачи.

$$l = 1; a = 1; \varphi[x\_] := 1 - \frac{x}{l};$$

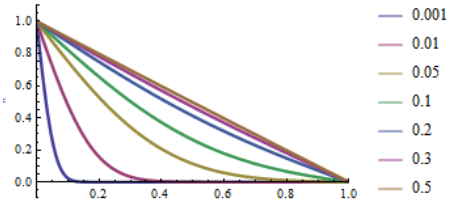
$$\theta 3m[\tau\_, x\_, t\_] := \text{EllipticTheta}\left[3, \pi\frac{\tau - x}{2l}, e^{-\frac{\pi^2 a^2 t}{l^2}}\right]$$

$$\theta 3p[\tau\_, x\_, t\_] := \text{EllipticTheta}\left[3, \pi\frac{\tau + x}{2l}, e^{-\frac{\pi^2 a^2 t}{l^2}}\right]$$

$$u[x\_, t\_] := 1 - \frac{x}{l} - \frac{1}{2l}\text{NIntegrate}[\varphi[\tau](\theta 3m[\tau, x, t] - \theta 3p[\tau, x, t]),$$
$$\{\tau, 0, l\}, \text{AccuracyGoal} \to 6, \text{PrecisionGoal} \to 6]$$

$$T = \{0.001, 0.01, 0.05, 0.1, 0.2, 0.3, 0.5\};$$

$$\text{Plot}[\{u[x, 0.001], u[x, 0.01], u[x, 0.05], u[x, 0.1], u[x, 0.2],$$
$$u[x, 0.3], u[x, 0.5]\}, \{x, 0, 1\}, \text{PlotRange} \to \{\{0, 1\}, \{-0.1, 1.1\}\},$$
$$\text{PlotStyle} \to \{\text{Thickness}[0.01]\}, \text{PlotLegends} \to T]$$



В моменты времени $t \geq 0.5$ графики распределения температуры практически не изменяются и совпадают с последним графиком. Это значит, что в стержне установилась температура $u(x,t) = 1 - x$ и начиная с момента t=0.5 влияние начального условия пренебрежимо мало. □

Решение задачи распространения тепла в неограниченном во всех направлениях двумерном теле, имеющего в нулевой момент времени температуру $u(x, y, 0) = \varphi(x, y)$, дается следующей формулой.

$$u(x, y, t) = \frac{1}{4\pi a^2 t}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\varphi(\xi, \eta)e^{-\frac{(x-\xi)^2 + (y-\eta)^2}{4a^2 t}}\, d\xi\, d\eta$$

Если начальная температура такова, что может быть представлена в виде $\varphi(x, y) = \varphi_1(x) \cdot \varphi_2(y)$, то этот интеграл преобразуется в произведение двух интегралов

$$u(x, y, t) = \frac{1}{4\pi a^2 t}\int_{-\infty}^{\infty}\varphi_1(\xi)e^{-\frac{(x-\xi)^2}{4a^2 t}}\, d\xi \cdot \int_{-\infty}^{\infty}\varphi_2(\eta)e^{-\frac{(y-\eta)^2}{4a^2 t}}\, d\eta$$

Если в них сделать замену $\tau = \frac{\xi - x}{2a\sqrt{t}}$ и $\nu = \frac{\eta - y}{2a\sqrt{t}}$, то мы получим

$$u(x, y, t) = \frac{1}{\sqrt{\pi}}\int_{-\infty}^{\infty}\varphi_1\left(x + 2a\sqrt{t}\,\tau\right)e^{-\tau^2}\, d\tau \times \frac{1}{\sqrt{\pi}}\int_{-\infty}^{\infty}\varphi_2\left(y + 2a\sqrt{t}\,\nu\right)e^{-\nu^2}\, d\nu \qquad (8)$$

Пример. *Бесконечная плоскость. Начальная температура равна нулю везде, кроме области квадрата* $D=[-1,1]\times[-1,1]$, *в которой она равна единице*.

Поскольку $\varphi(x,y)=\varphi_1(x)\cdot\varphi_2(y)$, где $\varphi_i(u)=\begin{cases}1, & -1\le u\le 1\\ 0, & |u|>1\end{cases}$ $(i=1,2)$, то решение может быть представлено в форме (8). Имеем

```
φ[x_] = Piecewise[{{0, x ≤ -1}, {1, x < 1}}, 0];
a =.;
u[x_, t_] = FullSimplify[1/Sqrt[π] Integrate[φ[x + 2a Sqrt[t] τ] Exp[-τ^2],
                {τ, -∞, ∞}, Assumptions :→ t > 0 && a > 0]]
```

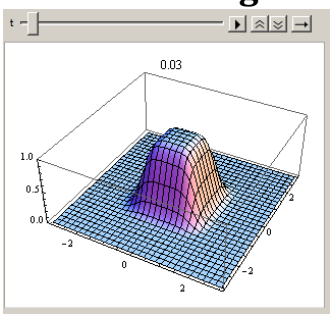
$$\frac{1}{2}\left(\mathrm{Erf}\left[\frac{1-x}{2a\sqrt{t}}\right]+\mathrm{Erf}\left[\frac{1+x}{2a\sqrt{t}}\right]\right)$$

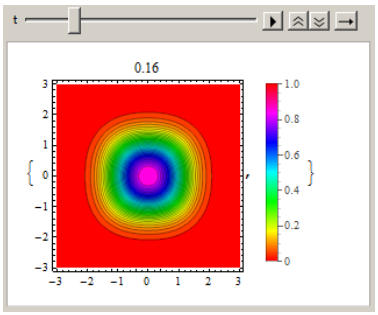
Теперь построим анимацию решения. Вначале представим решение как поверхность (панель анимации показана на следующем рисунке слева).

```
a = 1; U[x_, y_, t_] := u[x, t] * u[y, t];
T = Sequence[Join[{0.001}, Table[0.01 * i, {i, 80}]]];
Animate[Plot3D[Evaluate[U[x, y, t]], {x, -3, 3}, {y, -3, 3},
                PlotPoints → 31, PlotRange → {0, 1}, Mesh → Full,
                PlotLabel → t], {t, T}, AnimationRunning → False]
```

Во второй панели анимации представим решение контурным графиком (панель анимации показана на следующем рисунке справа).

```
Animate[{ContourPlot[U[x, y, t], {x, -3, 3}, {y, -3, 3}, Contours → 31,
                ColorFunction → Hue, ColorFunctionScaling → False,
                PlotPoints → 31, PlotRange → Full, PlotLabel → t],
                BarLegend[{(Hue[#]&), {0, 1}}]},
  {t, T}, AnimationRunning → False]
```





3.8.3 Одномерные колебания

Поперечные колебания струны, продольные колебания упругого стержня и многие другие явления описываются одномерным волновым уравнением

$$u''_{tt}=a^2u''_{xx}+f(x,t) \qquad (1)$$

Для однозначного определения решения необходимо задать начальные условия

$$u(x,0)=\varphi(x),\ u'_t(x,t)=\psi(x) \qquad (2)$$

и граничные условия, если у колеблющегося объекта имеются границы.

Известно общее решение Даламбера задачи (1), (2) для неограниченной прямой [2]

$$u(x,t)=\frac{\varphi(x+at)+\varphi(x-at)}{2}+\frac{1}{2a}\int_{x-at}^{x+at}\psi(\alpha)d\alpha+\frac{1}{2a}\int_0^t\int_{x-a(t-\tau)}^{x+a(t-\tau)}f(\xi,\tau)d\xi\ d\tau \qquad (3)$$

$(-\infty<x<\infty, t\ge 0)$. Формула (3) при любых функциях $\varphi(x), \psi(x)$ и $f(x,t)$ дает решение уравнения (1).

Основная идея построения интегрального решения первой краевой задачи для одномерного волнового уравнения (1) на конечном отрезке [*0,L*] при начальных условиях (2) и нулевых граничных условиях $u(0,t)=0,\ u(L,t)=0\quad t\ge 0$ состоит в следующем. В соответствии с методом продолжения выполняется нечетное периодическое продолжение начальных функций $\varphi(x), \psi(x)$ и $f(x,t)$ на всю ось О*x* [8]. Подстановка продолженных таким образом функций в (3) дает функцию *u*(*x*,*t*), являющуюся решением уравнения колебаний (1), удовлетворяющую на отрезке $[0, L]$ начальным условиям (2) и обращающуюся в ноль на концах этого отрезка. Известно [8, 9], что результирующая формула может быть представлена в виде

$$u(x,t)=\frac{1}{2}\left((-1)^{\left[\frac{x+at}{L}\right]}\varphi(stc(x+at,2L))+(-1)^{\left[\frac{x-at}{L}\right]}\varphi(stc(x-at,2L))\right)+$$

$$+\frac{1}{2a}\int_{stc(x-at,2L)}^{stc(x+at,2L)}\psi(\alpha)d\alpha+\frac{1}{2a}\int_0^t d\tau\int_{stc(x-a(t-\tau),2L)}^{stc(x+a(t-\tau),2L)}f(\xi,\tau)d\xi\quad (0\le x\le L, t\ge 0) \qquad (4)$$

где квадратные скобки [*z*] используются для обозначения целой части числа *z*, а функция $stc(x,w)$ представляет периодическую пилообразную непрерывную кусочно-линейную функцию, определяемую формулой (2.6). Приведем ее здесь еще раз.

$$stx(x,w)=\left|x-w\left[\frac{x}{w}+\frac{1}{2}\right]\right| \quad \text{или} \quad stc(x,w)=\frac{w}{2\pi}\arccos\left(\cos\frac{2\pi x}{w}\right), \qquad (5)$$

В части 2 п.2.4.5 мы рассмотрели несколько примеров, в которых полагали начальную скорость равной нулю $u'_t(x,t)=\psi(x)=0$ и $f(x,t)=0$. Если положить начальное смещение и внешнее усилие равными нулю $\varphi(x)=0$ и $f(x,t)=0$, то решение примет вид

$$u(x,t)=\frac{1}{2a}\int_{stc(x-at,2L)}^{stc(x+at,2L)}\psi(\alpha)d\alpha \qquad (0\le x\le L, t\ge 0). \qquad (6)$$

Во всех примерах этого пункта будем полагать скорость распространения колебаний равной единице $a=1$.

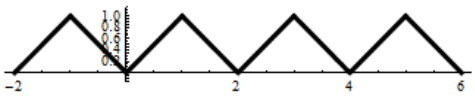
Пример. *Конечная струна длины L=1, закрепленная на концах. Начальное смещение ноль. Начальная скорость в форме ступеньки.* Струна возбуждается ударом жесткого плоского молоточка, сообщающего ей начальное распределение скоростей

$$\left.\frac{\partial u}{\partial t}\right|_{t=0} = \psi(x) = \begin{cases} 0, & 0 \le x < c-\delta \\ v_0, & c-\delta \le x \le c+\delta \\ 0, & c+\delta < x \le L \end{cases},$$

где c – координата середины точки удара, δ – полуширина ступеньки.

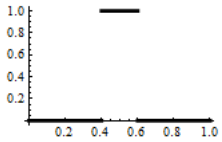
Пусть начальная скорость всюду ноль, кроме отрезка [0.4, 0.6], где она равна единице, т.е. $u'_t(x,0) = \psi(x) = \begin{cases} 1, 0.4 \le x \le 0.6 \\ 0, x < 0.4 \vee x > 0.6 \end{cases}$. Решение будем строить, используя формулу (6). Определим функцию $stc(x,w)$.

```
stc[x_, w_] = Abs[x - w Floor[x/w + 1/2]];
Plot[stc[x, 2], {x, -2, 6}, AspectRatio → Automatic]
```



Создадим функцию начальной скорости $\psi(x)$.

```
ψ[x_] = Piecewise[{{0, x ≤ 0.4}, {1, x < 0.6}}, 0];
Plot[ψ[x], {x, 0, 1}]
```

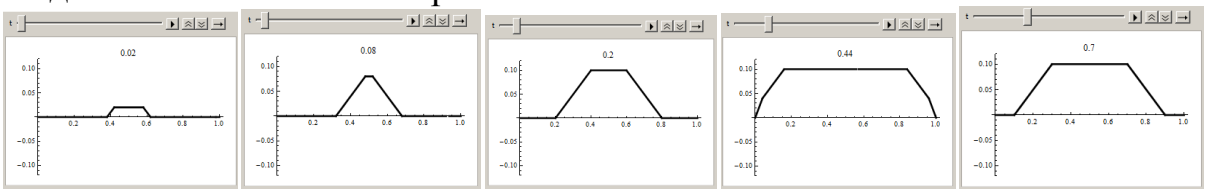


Создаем функцию решения и строим анимацию.

```
a =.; v[x_, t_] = 1/(2a) Simplify[Integrate[ψ[τ], {τ, α, β},
                                  Assumptions :→ α > 0 && β > 0]]
u[x_, t_] = v[x, t]/.{α → stc[x - at, 2L], β-> stc[x + at, 2L]};
a = 1; L = 1; T = Range[0, 1.98, 0.02];

Animate[
    Plot[u[x, t], {x, 0, 1}, PlotRange → {-0.12, 0.12}, PlotLabel → t],
{t, T}, AnimationRunning → False]
```

На следующем рисунке показаны профили струны в некоторые последовательные моменты времени.



Для этой краевой задачи в литературе приводится решение в виде ряда Фурье ([2], стр. 156-157)

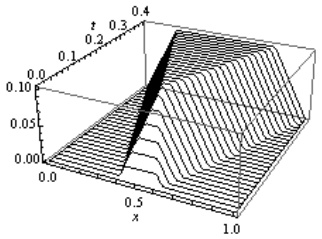
$$u(x,t) = \frac{4v_0 L}{\pi^2 a}\sum_{k=1}^{\infty}\frac{1}{k^2}\sin\frac{k\pi c}{L}\sin\frac{k\pi\delta}{L}\sin\frac{k\pi x}{L}\sin\frac{k\pi a t}{L}$$

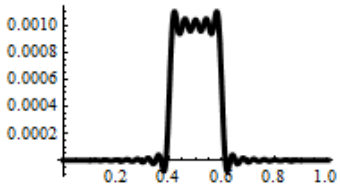
Вод код для построения формы струны с помощью ряда.

```
a = 1; L = 1; v0 = 1; δ = 0.1; c = 0.5;
```

$$\mathbf{uf[x\_,t\_]=\frac{4v0L}{\pi^2 a}Sum[\frac{1}{k^2}Sin[\frac{k\pi c}{L}]Sin[\frac{k\pi\delta}{L}]Sin[\frac{k\pi x}{L}]Sin[\frac{k\pi a t}{L}],\{k,1,50\}];}$$

$$\mathbf{Plot3D[uf[x,t],\{x,0,L\},\{t,0,0.4\},Mesh\to\{0,25\},PlotPoints\to 40,}$$
$$\mathbf{PlotStyle\to None,AxesLabel\to\{x,t\}]}$$





На предыдущем рисунке слева представлены профили струны в дискретные моменты времени $0\le t\le 0.4$ (период колебаний равен 2). Как видим, решения полученные обоими способами, совпадают. Но во втором решении мы ограничили сумму 50 – ю слагаемыми, и поэтому оно является приближенным. Если построить график решения в близкий к нулевому момент времени, то осцилляции, создаваемые частичной суммой ряда Фурье, будут заметны.

$$\mathbf{Plot[uf[x,0.001],\{x,0,1\}]}$$ (* предыдущий рисунок справа *)

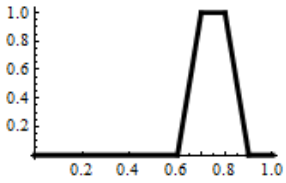
Это связано с тем, что точное (обобщенное) решение принадлежит классу C^0, а не C^∞, которому принадлежит частичная сумма ряда Фурье.

□

Пример. *Конечная струна, закрепленная на концах. Начальное смещение ноль. Внешнее усилие ноль. Начальная скорость в форме трапеции.*

Создаем функцию $\psi(x)$.

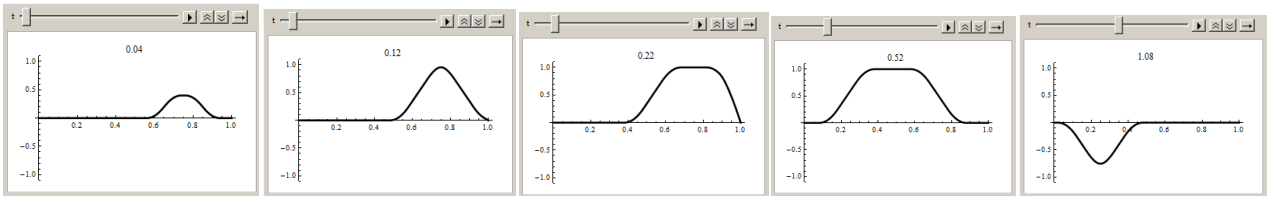
$$\mathbf{Clear[\alpha,\beta,a,L,x,w,\psi,v,u];}$$
$$\mathbf{\psi[x\_]=10\ Piecewise[\{\{0,x\le 0.6\},\{x-0.6,x<0.7\},\{0.1,x\le 0.8\},\{0.9-x,x\le 0.9\}\},0];}$$
$$\mathbf{Plot[\psi[x],\{x,0,1\}]}$$



Поскольку функция `Integrate` умеет работать с кусочными функциями, то при построении решения будем использовать ее (а не `NIntegrate`).

$$\mathbf{v[x\_,t\_]=\frac{1}{2a}Simplify[Integrate[\psi[\tau],\{\tau,\alpha,\beta\},Assumptions:\to \alpha>0\ \&\&\ \beta>0]];}$$
$$\mathbf{u[x\_,t\_]=v[x,t]/.\{\alpha\to stc[x-at,2L],\beta\to stc[x+at,2L]\};}$$
$$\mathbf{a=1;L=1;T=Range[0,1.98,0.02];}$$
$$\mathbf{Animate[}$$
$$\mathbf{Plot[u[x,t],\{x,0,1\},PlotRange\to\{-0.12,0.12\},PlotLabel\to t],}$$
$$\mathbf{\{t,T\},AnimationRunning\to False]}$$

На следующем рисунке показана панель анимации в некоторые последовательные моменты времени.



Здесь мы использовали код, создающий функцию `stc`, из предыдущего примера. Заметим, что при создании функции решения $u(x,t)$ можно было бы использовать функцию `NIntegrate`, например так

$$\mathbf{u[x\_,t\_]} := (\boldsymbol{\alpha} = \mathbf{stc}[\boldsymbol{x} - \boldsymbol{at}, \mathbf{2}\boldsymbol{L}]; \boldsymbol{\beta} = \mathbf{stc}[\boldsymbol{x} + \boldsymbol{at}, \mathbf{2}\boldsymbol{L}];$$
$$\frac{\mathbf{1}}{\mathbf{2}\boldsymbol{a}}\mathbf{NIntegrate}[\boldsymbol{\psi}[\boldsymbol{\tau}], \{\boldsymbol{\tau}, \boldsymbol{\alpha}, \boldsymbol{\beta}\}]);$$

Однако в этом случае вычисления будут выполняться значительно дольше.

□

При решении одномерного волнового уравнения на конечном отрезке часто используется метод разделения переменных. Рассмотрим, к каким формулам приводит этот метод при решении 1 – й краевой задачи для конечной струны.

Пусть $f(x,t)=0$, начальные условия имеют вид (2)

$$u(x,0)=\varphi(x),\ u'_t(x,t)=\psi(x)\ \ (0\le x\le L), \tag{2}$$

граничные условия равны 0 ($u(0,t)=u(L,t)=0$) и согласованы с начальными условиями, т.е. $\varphi(0)=\varphi(L)=0$. Разложим функции $\varphi(x),\psi(x)$ на отрезке [0,L] в ряд по синусам.

$$\varphi(x)=\sum_{n=1}^{\infty}\alpha_n\sin\frac{n\pi x}{L},\ \text{где}\ \alpha_n=\frac{2}{L}\int_0^L\varphi(x)\sin\frac{n\pi x}{L}dx$$

$$\psi(x)=\sum_{n=1}^{\infty}\beta_n\sin\frac{n\pi x}{L},\ \text{где}\ \beta_n=\frac{2}{L}\int_0^L\psi(x)\sin\frac{n\pi x}{L}dx$$

Тогда функция

$$u(x,t)=\sum_{n=1}^{\infty}\alpha_n\sin\frac{n\pi x}{L}\cos\frac{n\pi at}{L}+\frac{\beta_n L}{n\pi a}\sin\frac{n\pi x}{L}\sin\frac{n\pi at}{L} \tag{7}$$

будет представлять решение уравнения (1), удовлетворять начальным условиям (2) и нулевым граничным условиям. То, что функция $u(x,t)$ удовлетворяет уравнению (1) и нулевым граничным условиям следует из того, что каждое слагаемое в (7) удовлетворяет уравнению и граничным условиям. Выполнение начальных условий проверяется подстановкой t=0 в выражение (7) и в ее производную по t. Действительно, из (7) при t=0 сразу получаем $u(x,0)=\sum_{n=1}^{\infty}\alpha_n\sin\frac{n\pi x}{L}=\varphi(x)$. Далее из (7) имеем

$$u'_t(x,0)=\sum_{n=1}^{\infty}\frac{\beta_n L}{n\pi a}\sin\frac{n\pi x}{L}\cos\frac{n\pi at}{L}\left(\frac{n\pi a}{L}\right)\Bigg|_{t=0}=\sum_{n=1}^{\infty}\beta_n\sin\frac{n\pi x}{L}=\psi(x)$$

Таким образом, алгоритм решения уравнения колебаний конечной струны с закрепленными концами и нулевым внешним усилием состоит в следующем. Надо разложить начальные функции $\varphi(x),\psi(x)$ на отрезке [0,L] в ряд по

синусам и, зная коэффициенты α_n, β_n построить ряд (7). Напишем общую процедуру решения уравнения колебаний конечной струны с закрепленными концами, реализующую формулу (7). Имеем

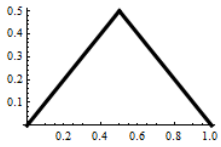
```
Clear[solFourier]
solFourier = Function[{x, t, a, L, φ, ψ, n}, Block[{fφ, fψ, gφ, gψ},
fφ = Chop[N[FourierSinSeries[φ[x], x, n,
                              FourierParameters → {1, π/L}]]];
fψ = Chop[N[FourierSinSeries[ψ[x], x, n,
                              FourierParameters → {1, π/L}]]];
gφ = fφ/.{Sin[w_] → Sin[w]Cos[w/x at]};
gψ = fψ/.{Sin[w_] → Sin[w]Sin[w/x at]/(w/x)};
gφ + gψ]];
```

Функция `solFourier` возвращает выражение частичной суммы (7), приближающей решение *u*(*x*,*t*). Она принимает следующие аргументы: идентификаторы/имена переменных *x* и *t*, значение параметра *a* из уравнения (1), длину отрезка *L*, функции начального смещения φ и начальной скорости ψ (не выражения), количество членов `n` частичной суммы Фурье. Протестируем созданную функцию.

Пример. *Конечная струна, закрепленная на концах. Начальная скорость 0. Начальное смещение в форме треугольника.*

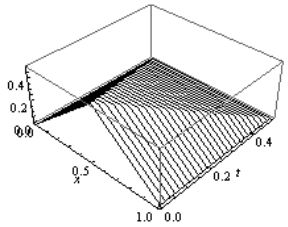
Создаем функцию смещения $\varphi(x)$.

```
φ[x_] = Piecewise[{{x, x ≤ 0.5}}, 1 - x];
Plot[φ[x], {x, 0, 1}]
```



Создаем функцию решения *u*(*x*,*t*) и строим график

```
u[x_, t_] = solFourier[x, t, 1, 1, φ, 0&,50];
Plot3D[u[x, t], {x, 0, 1}, {t, 0, 0.5}, Mesh → {0, 25}, PlotPoints → 40,
                                PlotStyle−> None, AxesLabel−> {x, t}]
```

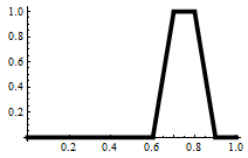


Обратите внимание на то, как мы задаем нулевую скорость – в виде чистой функции 0&.

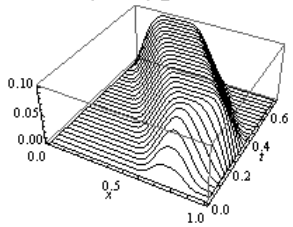
Сравните с решением этой задачи в п.2.4.5 (пример 5.4) предыдущей части. Решения идентичны, если не обращать внимание на то, что частичная сумма представляет приближенное решение. Выбор количества членов ряда равное 50 делает погрешность вычислений малой.

Пример. *Конечная струна, закрепленная на концах. Начальное смещение ноль. Начальная скорость в форме трапеции.*

$\boldsymbol{\psi}[\mathrm{x\_}] = \mathbf{10Piecewise}[\{\{\mathbf{0}, \boldsymbol{x} \leq \mathbf{0.6}\}, \{\boldsymbol{x} - \mathbf{0.6}, \boldsymbol{x} < 0.7\}, \{\mathbf{0.1}, \boldsymbol{x} \leq \mathbf{0.8}\},$
$\{\mathbf{0.9} - \boldsymbol{x}, \boldsymbol{x} \leq \mathbf{0.9}\}\}, \mathbf{0}];$
$\mathbf{Plot}[\boldsymbol{\psi}[\boldsymbol{x}], \{\boldsymbol{x}, \mathbf{0}, \mathbf{1}\}, \mathbf{PlotStyle} \rightarrow \{\mathbf{Black}, \mathbf{Thickness}[\mathbf{0.02}]\}]$



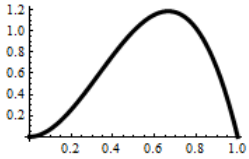
$\boldsymbol{u}[\mathrm{x\_}, \mathrm{t\_}] = \mathbf{solFourier}[\boldsymbol{x}, \boldsymbol{t}, \mathbf{1}, \mathbf{1}, \mathbf{0}\&, \boldsymbol{\psi}, \mathbf{50}];$
$\mathbf{Plot3D}[\boldsymbol{u}[\boldsymbol{x}, \boldsymbol{t}], \{\boldsymbol{x}, \mathbf{0}, \mathbf{1}\}, \{\boldsymbol{t}, \mathbf{0}, \mathbf{0.7}\}, \mathbf{Mesh} \rightarrow \{\mathbf{0}, \mathbf{25}\}, \mathbf{PlotPoints} \rightarrow \mathbf{40},$
$\mathbf{PlotStyle} -> \mathbf{None}, \mathbf{AxesLabel} -> \{\boldsymbol{x}, \boldsymbol{t}\}]$



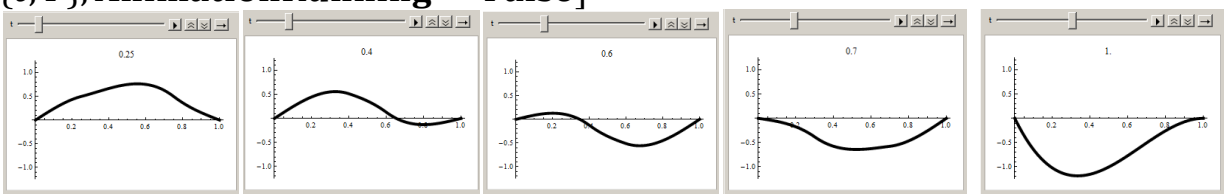
Сравните с решением этой задачи, полученной нами в этом пункте ранее. Решения идентичны.

Пример. *Конечная струна, закрепленная на концах. Начальная скорость 0. Начальное смещение в форме кубической параболы.*

$\boldsymbol{\varphi}[\mathrm{x\_}] = \mathbf{8}\, \boldsymbol{x}^{\mathbf{2}}(\mathbf{1} - \boldsymbol{x});$
$\mathbf{Plot}[\boldsymbol{\varphi}[\boldsymbol{x}], \{\boldsymbol{x}, \mathbf{0}, \mathbf{1}\}, \mathbf{PlotStyle} \rightarrow \{\mathbf{Black}, \mathbf{Thickness}[\mathbf{0.02}]\}]$



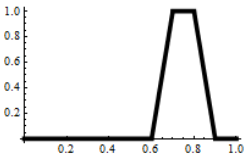
$\boldsymbol{L} = \mathbf{1}; \boldsymbol{u}[\mathrm{x\_}, \mathrm{t\_}] = \mathbf{solFourier}[\boldsymbol{x}, \boldsymbol{t}, \mathbf{1}, \boldsymbol{L}, \boldsymbol{\varphi}, \mathbf{0}\&, 50];$
$\boldsymbol{T} = \mathbf{Range}[\mathbf{0}, \mathbf{1.975}, \mathbf{.025}];$
$\mathbf{Animate}[\mathbf{Plot}[\boldsymbol{u}[\boldsymbol{x}, \boldsymbol{t}], \{\boldsymbol{x}, \mathbf{0}, \boldsymbol{L}\}, \mathbf{PlotStyle} \rightarrow \{\mathbf{Black}, \mathbf{Thickness}[\mathbf{0.015}]\},$
$\mathbf{PlotRange} \rightarrow \{-\mathbf{1.25}, \mathbf{1.25}\}, \mathbf{PlotLabel} \rightarrow \boldsymbol{t}],$
$\{\boldsymbol{t}, \boldsymbol{T}\}, \mathbf{AnimationRunning} \rightarrow \mathbf{False}]$



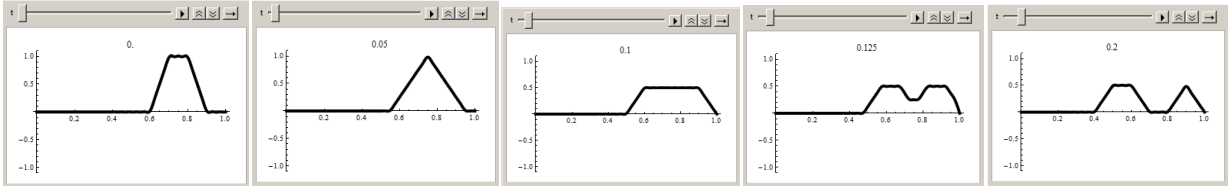
На рисунке показана панель анимации в моменты времени t = 0.25, 0.4, 0.6, 0.7, 1.0.

Пример. *Конечная струна, закрепленная на концах. Начальная скорость 0. Начальное смещение в форме трапеции.*

$\boldsymbol{\varphi}[\mathrm{x\_}] = \mathbf{10Piecewise}[\{\{\mathbf{0}, \boldsymbol{x} \leq \mathbf{0.6}\}, \{\boldsymbol{x} - \mathbf{0.6}, \boldsymbol{x} < 0.7\}, \{\mathbf{0.1}, \boldsymbol{x} \leq \mathbf{0.8}\},$
$\{\mathbf{0.9} - \boldsymbol{x}, \boldsymbol{x} \leq \mathbf{0.9}\}\}, \mathbf{0}];$
$\mathbf{Plot}[\boldsymbol{\psi}[\boldsymbol{x}], \{\boldsymbol{x}, \mathbf{0}, \mathbf{1}\}, \mathbf{PlotStyle} \rightarrow \{\mathbf{Black}, \mathbf{Thickness}[\mathbf{0.02}]\}]$



```
L = 1; u[x_, t_] = solFourier[x, t, 1, L, φ, 0&, 50];
T = Range[0, 1.975, .025];
Animate[Plot[u[x, t], {x, 0, L}, PlotStyle → {Black, Thickness[0.015]},
        PlotRange → {−1.1, 1.1}, PlotLabel → t],
{t, T}, AnimationRunning → False]
```



На рисунке показана панель анимации в моменты времени $t = 0, 0.05, 0.1, 0.125, 0.2$. Обратите внимание на кадр в момент времени t=0. На графике видна некоторая рябь, что вызвано недостаточной точностью приближения решения частичной суммой с 50 – ю членами.

Выше мы рассмотрели несколько примеров, в которых полагали внешнее усилие равным нулю $f(x,t)=0$. Если положить начальное смещение и скорость равным нулю $\varphi(x)=0$ и $\psi(x)=0$, то для конечного отрезка решение (4) примет вид

$$u(x,t)=\frac{1}{2a}\int_0^t d\tau \int_{stc(x-a(t-\tau),2L)}^{stc(x+a(t-\tau),2L)} f(\xi,\tau)d\xi \quad (-\infty<x<\infty, t\ge 0) \tag{8}$$

Рассмотрим несколько примеров применения этой формулы.

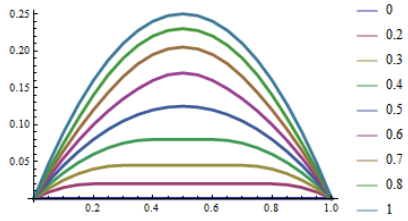
Пример. *Конечная струна, закрепленная на концах. Начальное смещение и скорость 0. Внешнее усилие постоянно.*

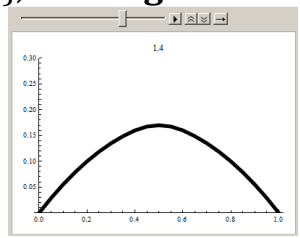
Струна находится в состоянии равновесия и в нулевой момент времени к ней прикладывается единичная сила, т.е. $\varphi(x)=0, \psi(x)=0, f(x,t)=1$. Начинается колебательный процесс. Решение может быть представлено по формуле (8). Внутренний интеграл вычисляется тривиально и задача сводится к однократному интегралу. Первообразную функции `stc` можно представить в символьном виде. Однако *Mathematica* это не умеет делать. Поэтому мы будем использовать численное интегрирование.

```
a = 1; L = 1;
u[x_, t_] := 1/(2a) NIntegrate[stc[x + a(t − τ), 2L] − stc[x − a(t − τ), 2L],
                {τ, 0, t}, AccuracyGoal → 4, PrecisionGoal → 4]
```

Численное интегрирование кусочных функций выполняется довольно долго. Для ускорения работы мы вычислим значения функции $u(x,t)$ в дискретном наборе точек и построим график решения в виде ломаной с небольшим шагом между точками по оси x. Графики решения в моменты времени $t=0, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 1.0$ показаны на следующем рисунке слева.

```
T = Join[{0}, Range[0.2, 0.8, 0.1], {1}];
U = Table[{x, u[x, t]}, {t, T}, {x, 0, 1, 0.05}];
ListLinePlot[U, PlotStyle → {Thickness[0.01]}, PlotLegends → T]
```





Для построения анимации добавим следующий код.

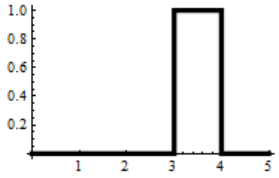
```
T = Range[0, 2, 0.05];
U = Table[{x, u[x, t]}, {t, T}, {x, 0, 1, 0.05}];
tp = Table[ListLinePlot[U[[i]], PlotRange → {0., 0.3},
          PlotStyle → {Black, Thickness[0.015]}, PlotLabel → T[[i]]],
    {i, Length[T] − 1}];
ListAnimate[tp, AnimationRunning → False]
```

Один из кадров анимации показан на предыдущем рисунке справа.

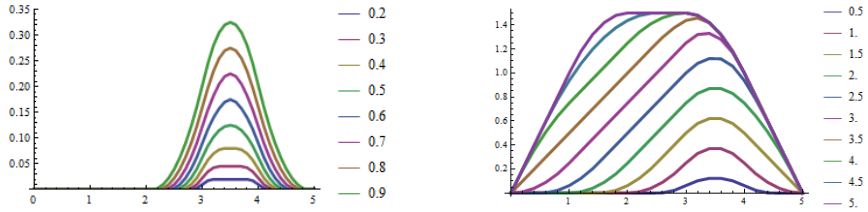
Пример. *Конечная струна, закрепленная на концах. Начальное смещение и скорость 0. Внешнее усилие не зависит от времени и имеет форму ступеньки.*

Струна длинной L=5 находится в состоянии равновесия $\varphi(x)=0, \psi(x)=0$ и в нулевой момент времени к ней прикладывается ступенчатая сила $f(x,t)=\begin{cases}1, 3\le x\le 4\\ 0, x<3\vee x>4\end{cases}$ $\forall\, t>0$. Начинается колебательный процесс. Решение строим по формуле (8), используя численное интегрирование.

```
Clear[α, β];  a = 1; L = 5;
stc[x_, w_] = Abs[x − wFloor[x/w + 1/2]];
f[x_] = UnitBox[x − 3.5];
Plot[f[x], {x, 0, 5}]   (* график внешнего усилия *)
```




```
F[α_, β_] = Integrate[f[t], {t, α, β},
                Assumptions :→ α >= 0  &&  β >= 0  &&  α < β]
G[x_, t_, τ_] = F[stc[x − a(t − τ), 2L], stc[x + a(t − τ), 2L]];
u[x_, t_] := 1/(2a) NIntegrate[G[x, t, τ], {τ, 0, t},
                AccuracyGoal → 4, PrecisionGoal → 4];
T1 = Range[0.2, 0.9, 0.1];
U1 = Table[{x, u[x, t]}, {t, T1}, {x, 0, L, 0.1}];
ListLinePlot[U1, PlotStyle → {Thickness[0.01]},
PlotLegends → T1, PlotRange → {0, 0.35}]  (* левый график на след. рис. *)
```




```
T2 = Range[0.5, 5, 0.5];
U2 = Table[{x, u[x, t]}, {t, T2}, {x, 0, L, 0.2}];
ListLinePlot[U2, PlotStyle → {Thickness[0.01]},
             PlotLegends → T2]      (* правый график на предыдущем рисунке *)
```

На левом графике показаны профили струны в начальные моменты времени t = 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9; на правом – в моменты времени t = 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0.

3.8.4 Примеры решения краевых задач для уравнение Лапласа

Много прикладных задач сводится к решению уравнения Лапласа $\Delta u = 0$. Для однозначного определения решения к уравнению следует добавить граничные условия и условия убывания на бесконечности, если область неограничена. К уравнению Лапласа приводятся задачи исследования движения идеальной жидкости, электростатики, задачи стационарной теплопроводности, статического прогиба мембран и многие другие. Функции, удовлетворяющие уравнению Лапласа называются гармоническими.

Решение уравнения Лапласа является сложной задачей и явных формульных решений известно очень мало. Здесь мы рассмотрим несколько примеров для которых решение уравнения Лапласа можно представить в интегральной форме.

Гармоническая функция в верхней полуплоскости определяется по формуле

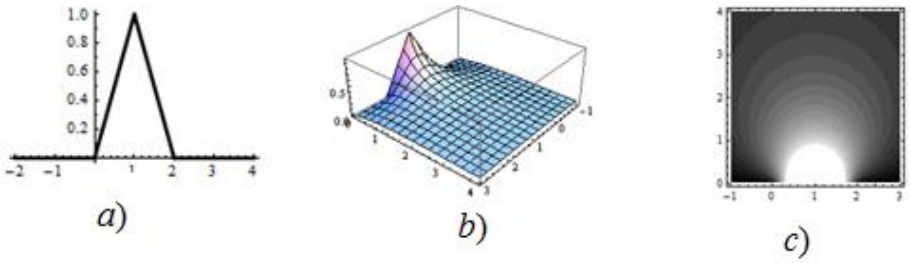
$$u(x,y) = \frac{1}{\pi}\int_{-\infty}^{\infty} u(\xi)\frac{y}{(x-\xi)^2 + y^2}d\xi\,, \qquad (1)$$

где $u(\xi)$ граничное значение функции. Если это граничное значение финитное (имеет конечный носитель – интервал на котором функция отлична от нуля), то интеграл можно вычислять по этому интервалу. Формулу (1) нельзя использовать при значении y=0, т.е. на границе области.

Пример. *Гармоническая функция в верхней полуплоскости. Граничное значение в форме треугольника*.

```
φ[x_] = Piecewise[{{0, x < 0}, {x, x < 1}, {2 - x, x < 2}}, 0];
Plot[φ[x], {x, -2, 4}]   (* следующий рисунок а *)
u[x_, y_] := 1/π (NIntegrate[ξ y/((x - ξ)^2 + y^2), {ξ, 0, 1}, WorkingPrecision → 5] +
                  NIntegrate[(2 - ξ) y/((x - ξ)^2 + y^2), {ξ, 1, 2}, WorkingPrecision → 5]);
```

```
Plot3D[u[x, y], {x, -1, 3}, {y, 0.01, 4}, PlotRange → All, PlotPoints → 40]
 ContourPlot[u[x, y], {x, -1, 3}, {y, 0.01, 4}, ContourLines → False,
          PlotPoints → 25, Contours → 20, ColorFunction → GrayLevel]
```



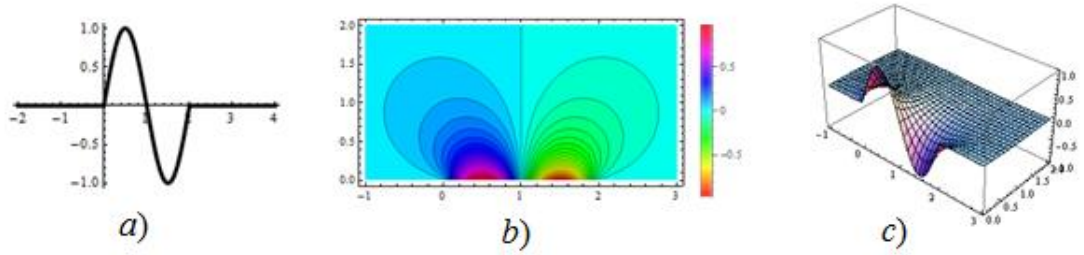
a) граничная функция $u(\xi)$; *b*) график/поверхность функции $u(x,y)$;
c) контурный график функции $u(x,y)$.

Пример. *Гармоническая функция в верхней полуплоскости. Граничное значение в форме одной волны синусоиды*.

```
φ[x_] = Piecewise[{{0, x < 0}, {Sin[πx], x < 2}}, 0];
Plot[φ[x], {x, -2, 4}]
u[x_, y_] := 1/π NIntegrate[φ[ξ] y / ((x - ξ)^2 + y^2), {ξ, 0, 2}, MaxRecursion → 100,
   PrecisionGoal → 4, AccuracyGoal → 4, Method -> "TrapezoidalRule"]
```

Поскольку численное интегрирование нашей функции выполняется довольно долго, то для ускорения работы мы вычислим значения функции $u(x,y)$ в дискретном наборе точек, который мы выбираем, а не в наборе точек, который выбирают сами графические функции. Для построения графиков решения по дискретному набору точек в пакете *Mathematica* имеются подходящие функции.

```
X = Range[-1, 3, 0.05];
Y = Join[{0.01}, Range[0.05, 2, 0.05]];
W = Flatten[Table[{x, y, u[x, y]}, {y, Y}, {x, X}], 1];
ListContourPlot[W, Contours → 40, ColorFunction → Hue,
   PlotRange → All, AspectRatio → Automatic, PlotLegends → Automatic]
ListPlot3D[W, PlotRange → All, BoxRatios → {4, 2, 2}, Mesh → {41, 16, 0}]
```



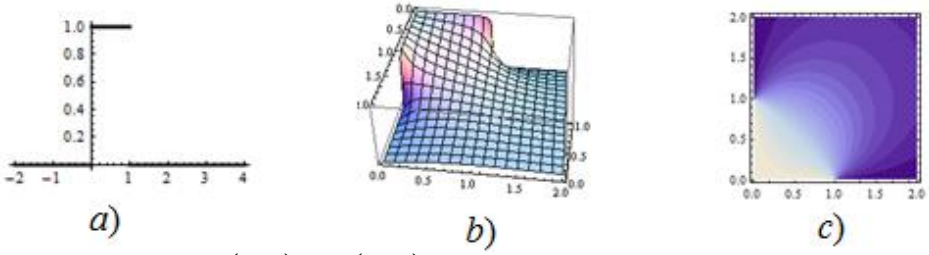
a) граничная функция $u(x,0)$; *b*) контурный график функции $u(x,y)$;
c) график/поверхность функции $u(x,y)$.

Используя метод продолжения, можно получить решение уравнения Лапласа в первом квадранте (в области $x \ge 0,\ y \ge 0$). Пусть граничные значения в 1-ом квадранте имеют вид $u(x,0) = \varphi(x),\ x \ge 0$, $u(0,y) = \psi(y),\ y \ge 0$. Решение уравнения Лапласа в 1-ом квадранте имеет вид

$$u(x,y)=\frac{4xy}{\pi}\int_0^{\infty}\left(\frac{\varphi(\xi)\xi}{((x-\xi)^2+y^2)((x+\xi)^2+y^2)}+\frac{\psi(\xi)\xi}{((y-\xi)^2+x^2)((y+\xi)^2+x^2)}\right)d\xi$$

Пример. *Гармоническая функция в первом квадранте. Граничное значение в форме прямоугольных столбиков на обеих полуосях X и Y.*

```
φ[x_] = Piecewise[{{0, x < 0}, {1, x < 1}}, 0];
Plot[φ[x], {x, -2, 4}]
u[x_, y_] := 4xy/π (NIntegrate[ξ φ[ξ]/((x - ξ)^2 + y^2)((x + ξ)^2 + y^2),
    {ξ, 0, 1}, MaxRecursion → 12, AccuracyGoal → 4,
    PrecisionGoal → 4] +
  NIntegrate[ξ φ[ξ]/((y - ξ)^2 + x^2)((y + ξ)^2 + x^2), {ξ, 0, 1},
    MaxRecursion → 12, AccuracyGoal → 4,
    PrecisionGoal → 4])
Plot3D[u[x, y], {x, 0.01, 2}, {y, 0.01, 2}, PlotRange → All, PlotPoints → 40]
ContourPlot[u[x, y], {x, 0.01, 2}, {y, 0.01, 2}, ContourLines → False,
  PlotPoints → 25, Contours → 20]
```



a) *b*) *c*)

a) граничная функция $u(\tau,0)=u(0,\tau)$; *b*) график/поверхность функции $u(x,y)$; *c*) контурный график функции $u(x,y)$.

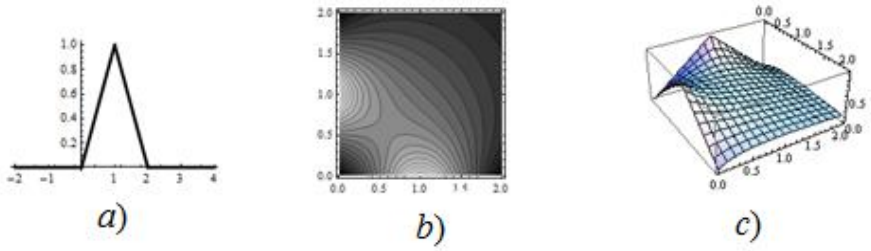
Пример. *Гармоническая функция в первом квадранте. Граничное значение в форме треугольников на обеих полуосях X и Y.*

```
φ[x_] = Piecewise[{{0, x < 0}, {x, x < 1}, {2 - x, x < 2}}, 0];
Plot[φ[x], {x, -2, 4}]
u[x_, y_] := 4xy/π (NIntegrate[ξφ[ξ]/((x - ξ)^2 + y^2)((x + ξ)^2 + y^2),
    {ξ, 0, 2}, AccuracyGoal → 4, PrecisionGoal → 4,
    MaxRecursion → 12] +
  NIntegrate[ξφ[ξ]/((y - ξ)^2 + x^2)((y + ξ)^2 + x^2), {ξ, 0, 2},
    AccuracyGoal → 4, PrecisionGoal → 4,
    MaxRecursion → 12])
```

Ускорим построение графиков вычислением значений функции в некотором наборе точек.

```
X = Join[{0.01}, Range[0.05, 2, 0.05]];
Y = Join[{0.01}, Range[0.05, 2, 0.05]];
W = Flatten[Table[{x, y, u[x, y]}, {y, Y}, {x, X}], 1];
```

ListContourPlot[W, **Contours** → **20**, **ColorFunction** → **GrayLevel**]
ListPlot3D[W]



a) *b*) *c*)

a) граничная функция $u(\tau,0)=u(0,\tau)$; *b*) контурный график функции $u(x,y)$;
c) график/поверхность функции $u(x,y)$.

Гармоническая в круге радиуса R функция вычисляется по формуле Пуассона

$$u(r,\varphi)=\frac{1}{2\pi}\int_0^{2\pi} f(\theta)\frac{R^2-r^2}{R^2-2Rr\cos(\theta-\varphi)+r^2}\,d\theta \qquad (2)$$

где $f(\theta)$ граничное значение функции, а r,φ - полярные координаты точки внутри круга.

Пример. *Стационарное распределение температуры внутри бесконечного кругового цилиндра. Температура на поверхности имеет вид косинусоиды.*

Ось цилиндра направим вдоль оси Oz. Температура точек u внутри цилиндра не зависит от z и будет удовлетворять двумерному уравнению Лапласа внутри круга радиуса R. Если использовать полярные координаты, то граничное условие для функции температуры $u(r,\varphi)$ будет иметь вид $u(R,\theta)=f(\theta)$. Сама функция $u(r,\varphi)$ будет определяться формулой (2).

Пусть $f(\theta)=\cos 3\theta$. Тогда

$$u(r,\varphi)=\frac{R^2-r^2}{2\pi}\int_0^{2\pi}\frac{\cos(3\theta)}{R^2-2Rr\cos(\theta-\varphi)+r^2}\,d\theta$$

Сделаем замену $\theta-\varphi=t$ и, учитывая периодичность (не нужно менять пределы интегрирования), получим

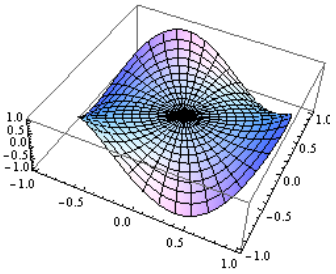
$$u(r,\varphi)=\frac{R^2-r^2}{2\pi}\int_0^{2\pi}\frac{\cos(3t+3\varphi)}{R^2-2Rr\cos t+r^2}\,dt$$

Тогда

```
R =.;
u[r_, φ_] = Simplify[(R^2 - r^2)/(2 π) Integrate[Cos[3 t + 3 φ]/(R^2 - 2 R r Cos[t] + r^2), {t, 0, 2 π},
                    Assumptions :→ (0 < r < R && φ ∈ Reals && R > 0)]]
```

$$\frac{r^3\cos[3\varphi]}{R^3}$$

```
R = 1;
RevolutionPlot3D[u[r, φ], {r, 0, 1}, {φ, 0, 2 π},
                 Mesh → {11, 41, 0}, PlotRange → All]
```



Замечание. Если замену $\theta-\varphi=t$ не делать, то *Mathematica* 9 возвращает неверный результат! Попробуйте.

```
Simplify[(R^2 - r^2)/(2 π) Integrate[Cos[3 t]/(R^2 - 2 R r Cos[t - φ] + r^2), {t, 0, 2 π},
    Assumptions :> (0 < r < R && φ ∈ Reals && R > 0)]]
```

$$\frac{(r^6-R^6)\cos[3\varphi]}{2r^3R^3}$$

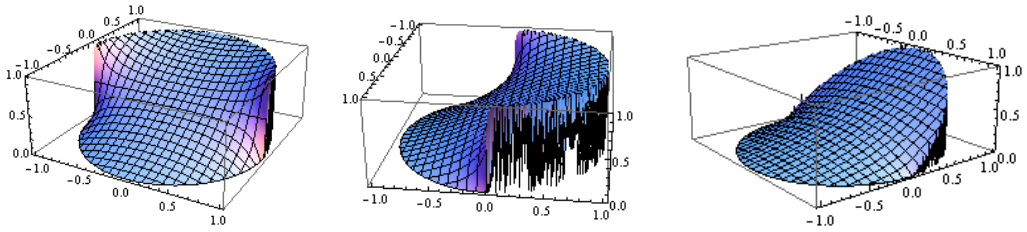
Пример. *Стационарное распределение температуры внутри бесконечного кругового цилиндра. Температура на поверхности кусочно постоянна.*

Пусть на поверхности бесконечного кругового цилиндра радиуса R поддерживается следующая температура: $u=T$ при $0<\varphi<\pi$; $u=0$ при $\pi<\varphi<2\pi$; (для любых *z* и *t*). Внутри цилиндра температура установилась и ее требуется найти. Задача сводится к решению двумерного уравнения Лапласа $\Delta u=0$ с граничным условием $u(R,\theta)=f(\theta)=\begin{cases}T, 0<\theta<\pi\\ 0, \pi<\theta<2\pi\end{cases}$. Решение дается интегралом Пуассона (2). Подставляя в него $f(\theta)=0$ при $\pi<\theta<2\pi$ и T при $0<\theta<\pi$, получим

$$u(x,y)=T\frac{R^2-x^2-y^2}{2\pi}\int_0^{\pi}\frac{1}{R^2-2R(x\cos\theta+y\sin\theta)+x^2+y^2}d\theta$$

Обратите внимание на замену $r\cos(\theta-\varphi)=x\cos\theta+y\sin\theta$, которую мы сделали в интеграле. Имеем

```
R = 1; T = 1;
u[x_, y_] := T (R^2 - x^2 - y^2)/(2 π) *
    NIntegrate[(1/(R^2 - 2 R (x Cos[θ] + y Sin[θ]) + x^2 + y^2)),
        {θ, 0, π}, AccuracyGoal → 4, PrecisionGoal → 4];
U[x_, y_] := If[x^2 + y^2 < R^2, u[x, y], 0];
Plot3D[U[x, y], {x, -R, R}, {y, -R, R}, PlotRange → {0, T},
    PlotPoints → 40, Mesh → 21,
    RegionFunction → Function[{x, y, z}, x^2 + y^2 < R^2]]
```



Если покрутить график, то вы заметите «рябь» у контура круга (рисунок в середине). Это вызвано тем, что интеграл Пуассона дает решение краевой задачи *внутри* круга, а граничное значение $f(\theta)$ он только приближает. Использовать его для вычисления значения функции $u(r,\varphi)$ на границе нельзя. В точках границы погрешность численного интегрирования вызывает «рябь». Даже если граничная функция $f(\theta)$ будет непрерывной, то «рябь» остается. Пусть, например, граничное значение имеет вид $f(\theta)=\begin{cases}\sin\theta\,,\ 0\le\theta<\pi\\ 0\,,\ \pi\le\theta<2\pi\end{cases}$. Тогда для построения графика решения в предыдущем коде надо изменить строку, определяющую функцию $u(x,y)$

```
u[x_, y_] := (R^2 - x^2 - y^2)/(2 π) NIntegrate[
                Sin[θ]/(R^2 - 2 R (x Cos[θ] + y Sin[θ]) + x^2 + y^2), {θ, 0, π},
                AccuracyGoal → 4, PrecisionGoal → 4];
```

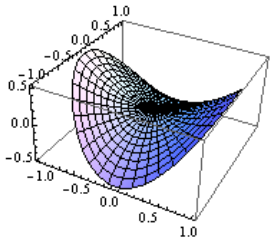
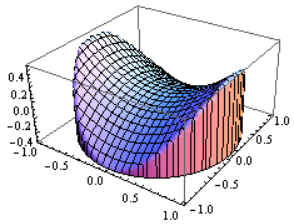
График решения показан на предыдущем рисунке справа..

Пример. *Тонкая мембрана натянута на проволочный каркас,* проектирующийся на плоскость Oxy в окружность радиуса R с центром в начале координат. Уравнение контура мембраны в цилиндрических координатах имеет вид: $u(R,\theta)=f(\theta)=h\cos 2\theta\ \ (0\le\theta<2\pi)$. Известно, что форма поверхности $z=u(x,y)$, по которой расположится пленка, удовлетворяет уравнению Лапласа и граничному условию $u(R,\theta)=f(\theta)$. Найдем эту форму, используя интеграл Пуассона (2). Прежде, чем вычислять интеграл, сделаем в нем замену $\theta-\varphi=t$ и учтем периодичность подынтегральной функции (не нужно менять пределы интегрирования). Имеем

```
R =.; h =.;
u[r_, φ_] = Simplify[h (R^2 - r^2)/(2 π)
              Integrate[Cos[2 t + 2 φ]/(R^2 - 2 R r Cos[t] + r^2), {t, 0, 2 π},
              Assumptions :→ (0 < r < R && φ ∈ Reals && R > 0)]]
```

$$\frac{h\,r^2\cos[2\varphi]}{R^2}$$

```
R = 1; h = 0.5;
RevolutionPlot3D[u[r, φ], {r, 0, 1}, {φ, 0, 2 π}, Mesh → {11, 41, 0},
        PlotRange → All, BoxRatios → {2 R, 2 R, 2 h}] (* след. рисунок слева *)
```

Делая замену $x = r\cos\varphi$, $y = r\sin\varphi$, переходим к представлению формы мембраны в декартовых координатах.

$$\boldsymbol{U}[\mathrm{x\_}, \mathrm{y\_}] = \boldsymbol{h}\frac{\boldsymbol{x^2 - y^2}}{\boldsymbol{R^2}};$$

$$\mathbf{RegionPlot3D}[\boldsymbol{z < U[x, y]}\ \&\&\ \boldsymbol{x^2 + y^2 < R^2},$$
$$\{\boldsymbol{x}, -\boldsymbol{R}, \boldsymbol{R}\}, \{\boldsymbol{y}, -\boldsymbol{R}, \boldsymbol{R}\}, \{\boldsymbol{z}, -\boldsymbol{h}, \boldsymbol{h}\}, \mathbf{PlotRange} \to \mathbf{All}, \mathbf{PlotPoints} \to \mathbf{41},$$
$$\mathbf{Mesh} \to \{\mathbf{21}, \mathbf{21}, \mathbf{0}\}, \mathbf{BoxRatios} \to \{\mathbf{2}\boldsymbol{R}, \mathbf{2}\boldsymbol{R}, \mathbf{2}\boldsymbol{h}\}] \text{ (* пред. рисунок справа *)}$$

Литература.

1. Очан Ю.С. Методы математической физики.
2. Н.С. Кошляков, Э.Б.Глинер, М.М.Смирнов Основные дифференциальные уравнения математической физики. ГИФМЛ.М. 1962, 767с.
3. Тихонов А.Н., Самарский А.А. Уравнения математической физики. – М.: Наука, 1977. – 736с.
4. Э.Т.Уиттекер, Дж.Н.Ватсон Курс современного анализа. Ч.2. ГИФМЛ. М. 1963г. 515с.
5. В.М. Бабич, М.Б. Капилевич, С.Г. Михлин и др. Под. ред. С.Г. Михлина. Справочная математическая библиотека. Линейные уравнения математической физики. – М.: Наука, 1964. – 368 с.
6. Полянин А.Д. Справочник по линейным уравнениям математической физики. М.: ФИЗМАТЛИТ, 2001. – 576с.
7. Карслоу Г., Егер Д. Теплопроводность твердых тел. – М.: Наука, 1964.– 487 с.
8. Доля П.Г. Периодическое продолжение функций и решение уравнения колебаний струны в системах символьной математики.// Вестник Харьк. нац. ун-та., - 2006.- № 733. Сер. ”Математическое моделирование.
9. Dolya P.G. Solution of the initial value problem for the inhomogeneous equation of vibrations of a finite string with homogeneous boundary conditions // Kharkiv: Verkin Institute for Low Temperature Physics and Engineering of NASU. Lapunov memorial conference., Book of abstracts, – p. 38-39, 2007

4. Решение дифференциальных уравнений в пакете Mathematica.

Многие прикладные задачи сводятся к решению обыкновенных дифференциальных уравнений (ОДУ) или их систем. Для некоторых уравнений и систем имеются формулы «точного» решения, и пакет *Mathematica* умеет их находить. В тех задачах, для которых символьное решение построить не удается, *Mathematica* может находить численное решение. В решениях она использует богатый набор специальных функций и свои интерполяционные функции, представляя решение в форме, в которой они могут быть непосредственно использованы. Данная часть пособия знакомит Вас с основными функциями системы, предназначенными для символьного и численного решения ОДУ и их систем, а также с вспомогательными функциями, используемыми при их исследовании.

4.1 Символьные решения ОДУ и их систем.

Система *Mathematica* обладает обширными возможностями решения обыкновенных дифференциальных уравнений и их систем в символьном виде. Для этого используется функция `DSolve`, в алгоритме которой реализовано большинство известных на сегодняшний день аналитических методов.

При поиске решения функция `DSolve` полагает, что имена искомых функций и имя независимой переменной не содержат значений в рабочем пространстве системы. Если это не так, то функция `DSolve` выдаст сообщение об ошибке. В таком случае имена переменных и функций, используемых `DSolve`, следует удалить из рабочего пространства командой `Remove[имя1,имя2,...]`. Все последующие примеры предполагают, что это уже сделано. Если все же у вас возникнут проблемы с выполнением примеров, то обратитесь в конец пособия в раздел «Замечания о выполнении примеров», где мы кратко описываем возможные трудности, с которыми вы можете столкнуться, и способы их устранения.

4.1.1 Решение обыкновенных дифференциальных уравнений.

Система *Mathematica* позволяет решать многие ОДУ в символьном виде с помощью функции `DSolve`. Функция имеет следующий синтаксис

```
DSolve[уравнение,y,x]
```

Она определяет функцию `y`, считая `x` независимой переменной уравнения. Первый аргумент – уравнение (или список уравнений), записанное в терминах функций и ее производных (`y[x]`, `y'[x]`, `y''[x]` и т.д.), но допустимо использовать другие способы обозначения производных (например, $D[y[x],x]$, $y^{(2)}[x]$, $\partial_x y[x]$, $\partial_{xx} y[x]$ и др.). Второй аргумент – имя искомой функции. Третий аргумент – независимая переменная. Результат возвращается в виде списка правил подстановки.

$\mathbf{DSolve}[y'[x] == x^2, y[x], x]$

$\{\{y[x] \to \frac{x^3}{3} + C[1]\}\}$

Здесь `C[1]` представляет произвольную константу интегрирования.

Функция `DSolve` в качестве первого аргумента может принимать список уравнений, среди которых могут быть и начальные условия.

$$\mathbf{DSolve}[\{\boldsymbol{D}[\boldsymbol{y}[\boldsymbol{x}],\boldsymbol{x}] == \boldsymbol{x}^{\mathbf{2}}, \boldsymbol{y}[\mathbf{0}] == \mathbf{1}\}, \boldsymbol{y}[\boldsymbol{x}], \boldsymbol{x}]$$

$$\{\{y[x] \to \frac{1}{3}(3 + x^3)\}\}$$

Она может также находить решение с произвольными параметрами в качестве коэффициентов.

$$\mathbf{DSolve}[\boldsymbol{y}'[\boldsymbol{x}] == \boldsymbol{a}\boldsymbol{y}[\boldsymbol{x}], \boldsymbol{y}[\boldsymbol{x}], \boldsymbol{x}]$$

$$\{\{y[x] \to e^{ax} C[1]\}\}$$

Символьные параметры могут присутствовать в качестве коэффициентов или значений в начальных условиях.

$$\mathbf{DSolve}[\{\boldsymbol{y}'[\boldsymbol{x}] == \boldsymbol{a}\boldsymbol{y}[\boldsymbol{x}], \boldsymbol{y}[\boldsymbol{c}] == \boldsymbol{b}\}, \boldsymbol{y}[\boldsymbol{x}], \boldsymbol{x}]$$

$$\{\{y[x] \to b e^{-ac+ax}\}\}$$

Функция `DSolve` ищет решение в символьном виде и пытается вернуть решение в квадратурах. Если уравнение содержит неопределенную функцию, то в решении появляются неопределенные интегралы.

$$\mathbf{DSolve}[\boldsymbol{y}'[\boldsymbol{x}] == \boldsymbol{f}[\boldsymbol{x}], \boldsymbol{y}[\boldsymbol{x}], \boldsymbol{x}]$$

$$\{\{y[x] \to C[1] + \int_1^x f[K[1]]\, dK[1]\}\}$$

Здесь K[1] обозначает переменную интегрирования, а `C[1]` представляет произвольную константу. Если такие обозначения вас не устраивает, то можно выполнить замену/подстановку следующим образом

$$\mathbf{DSolve}[\boldsymbol{y}'[\boldsymbol{x}] - \boldsymbol{y}[\boldsymbol{x}] == \boldsymbol{f}[\boldsymbol{x}], \boldsymbol{y}[\boldsymbol{x}], \boldsymbol{x}]/.\{\boldsymbol{K}[\mathbf{1}] \to \boldsymbol{t}, \boldsymbol{C}[\mathbf{1}] \to \boldsymbol{C_1}\}$$

$$\{\{y[x] \to e^x \int_1^x e^{-t} f[t]\, dt + e^x C_1\}\}$$

Для ввода C_1 наберите C, затем комбинацию Ctrl-_. В полученный шаблон  введите индекс 1. Для возврата на нормальный уровень ввода наберите комбинацию Ctrl – пробел. Запись **`/. имя1->имя2`** обозначает подстановку (символы **`/.`**) в предшествующее выражение вместо каждого вхождения набора символов **`имя1`** другого набора символов **`имя2`**.

Функция `DSolve` умеет работать с некоторыми разрывными функциями, например

$$\mathbf{DSolve}[\boldsymbol{D}[\boldsymbol{y}[\boldsymbol{x}], \{\boldsymbol{x}, \mathbf{2}\}] - \boldsymbol{y}[\boldsymbol{x}] == \mathbf{UnitStep}[\boldsymbol{x}], \boldsymbol{y}[\boldsymbol{x}], \boldsymbol{x}]$$

$$\{\{y[x] \to e^x C[1] + e^{-x} C[2] + \frac{1}{2} e^{-x}(-1 + e^x)^2 \text{UnitStep}[x]\}\}$$

Здесь `UnitStep` – функция «единичная ступенька»

$$UnitStep(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$$

Однако с задачей Коши для ДУ с разрывными функциями *Mathematica* иногда не справляется

$$\mathbf{DSolve}[\{\boldsymbol{y}'[\boldsymbol{x}] == \mathbf{UnitStep}[\boldsymbol{x}], \boldsymbol{y}[\mathbf{1}] == \mathbf{1}\}, \boldsymbol{y}[\boldsymbol{x}], \boldsymbol{x}]$$

$$\{\{y[x] \to (0/.\,\$\text{Aborted}[]) + (x - (0/.\,\$\text{Aborted}[]))\text{UnitStep}[x]\}\}$$

или справляется

$s = \mathbf{DSolve}[\{y'[x] + \mathbf{Max}[x, 1]y[x] == 0, y[0] == 1\}, y[x], x]$

$$\left\{\left\{y[x] \to \begin{cases} e^{-x} & x \le 1 \\ e^{-\frac{1}{2}-\frac{x^2}{2}} & \text{True} \end{cases}\right\}\right\}$$

Вместо имени константы интегрирования, выбираемой по-умолчанию, можно определить другой символ

$\mathbf{DSolve}[y'[x] == ay[x], y[x], x, \mathbf{GeneratedParameters} \to K]$

$\mathbf{DSolve}[y'[x] == ay[x], y[x], x, \mathbf{DSolveConstants} \to K]$ (в старых версиях)

$\{\{y[x] \to e^{ax} K[1]\}\}$

Вместо имени константы можно использовать функции, которым в качестве аргумента будут передаваться индексы (номера) постоянных. Удобно использовать функцию Subscript$[a, n]$, которая возвращает запись вида a_n.

$\mathbf{DSolve}[y''[x] == y[x], y[x], x, \mathbf{GeneratedParameters} \to (\mathbf{Subscript}[C, \#]\&)]$

$\{\{y[x] \to e^{x} C_1 + e^{-x} C_2\}\}$

В некоторых случаях *Mathematica* возвращает несколько решений:

$\mathbf{DSolve}[(\partial_x y[x])^2 + (y[x])^2 == 1, y[x], x]$ (решаем уравнение $(y')^2 + y^2 = 1$)

$\{\{y[x] \to -\text{Sin}[x - C[1]]\}, \{y[x] \to \text{Sin}[x + C[1]]\}\}$

Заметим, что здесь потеряны два решения $y(x) = \pm 1$ (в Mathematica 9.0).

Чтобы набрать выражение $\partial_x y[x]$ вы можете набрать комбинацию $Esc - dt - Esc$. Она создает шаблон ввода $\partial_{\square}\square$, который вы должны заполнить $\partial_x y[x]$. Можно также ввести комбинацию $Esc - pd - Esc$, которая создает символ ∂, затем комбинацию *Ctrl-_* (подчеркивание), затем *x* (появится ∂_x), затем → (клавиша управления текстовым курсором), затем введите *y*[*x*].

Однако, привычнее использовать «'» (штрих) для обозначения производных функции в ОДУ.

$\mathbf{DSolve}[y[x]y'[x] == 1, y, x]$

$\{\{y \to \text{Function}[\{x\}, -\sqrt{2}\sqrt{x + C[1]}]\}, \{y \to \text{Function}[\{x\}, \sqrt{2}\sqrt{x + C[1]}]\}\}$

`DSolve` может решать широкий спектр дифференциальных уравнений, и решение, обычно, получается в виде правила подстановок.

$\mathbf{DSolve}[y'[x] + y[x] == 1, y[x], x]$

$\{\{y[x] \to 1 + e^{-x} C[1]\}\}$

Здесь второй параметр функции `DSolve` был `y[x]`, а не просто `y` как в предыдущем примере. Допустимо использовать обе формы, однако между ними имеется различие. В последнем примере решение получается в виде правил для вычисления *выражения* `y[x]` (не *функции* и не ее частных значений, таких как `y[0]`). Например:

$y[x] + 2y'[x] + y[0]/.\%$

$\{1 + e^{-x} C[1] + y[0] + 2y'[x]\}$

Обратите внимание, что `y[0]` и `y'[x]` не были вычислены. Значки (символы) **/.** обозначают подстановку в предшествующее выражение результата последнего вычисления (знак `%`).

Вы можете попросить `DSolve` определить *функцию* `y` (а не *выражение* `y[x]`) и она вернет правило для вычисления функции `y`.

$\mathbf{DSolve}[\boldsymbol{y}'[\boldsymbol{x}] + \boldsymbol{y}[\boldsymbol{x}] == \mathbf{1}, \boldsymbol{y}, \boldsymbol{x}]$

$\{\{y \to \text{Function}[\{x\}, 1 + e^{-x} C[1]]\}\}$

В некоторых старых версиях результат будет возвращен в виде

```
{{y->(1+E^-#1 C[1]&)}}
```

Здесь символ #1 обозначает 1 – й аргумент будущей функции, а знак амперсанда (&) постфиксное определение функции. Напомним, что формальные параметры при определении функции обозначаются # (или #1, #2 и т.д.), само имя функции обозначается `#0`, `##` используется для обозначения всего списка аргументов, `##n` обозначает списка аргументов начиная с n-того.

Последнее выражение можно прочитать так: везде вместо имени функции `y` следует подставлять выражение `(1 + E^-#1 C[1])`, которое определяет функцию и, где C[1] символьная константа, #1 - имя первого (и единственного) аргумента функции. Полученное правило подстановки будет применяться везде, где имя функции `y` будет сочетаться с аргументам в квадратных скобках, например, `y[x]` или `y[0]`.

Теперь в выражении

$\boldsymbol{y}[\boldsymbol{x}] + \mathbf{2}\boldsymbol{y}'[\boldsymbol{x}] + \boldsymbol{y}[\mathbf{0}]/.\%$

правило подстановки будет применяться и к начальному значению `y[0]` и к производной `y'[x]`:

$\{2 + C[1] - e^{-x} C[1]\}$

Подстановка решения в уравнение в виде функции дает `True`.

$\boldsymbol{y}'[\boldsymbol{x}] + \boldsymbol{y}[\boldsymbol{x}] == \mathbf{1}/.\%\%$

```
{True}
```

Здесь символ `%%` (два процента) означает решение, полученное два шага назад. Итак,

`DSolve[eqn,y[x],x]` решает ДУ относительно *выражения* `y[x]`.

`DSolve[eqn,y,x]` решает ДУ относительно *функции* `y`.

Приведем еще несколько примеров.

$\mathbf{DSolve}[\boldsymbol{y}''[\boldsymbol{x}] + \boldsymbol{y}[\boldsymbol{x}] == \mathbf{0}, \boldsymbol{y}[\boldsymbol{x}], \boldsymbol{x}, \mathbf{GeneratedParameters} \to \text{Subscript}[\boldsymbol{C}, \#]\&)]$

$\{\{y[x] \to \text{Cos}[x] C_1 + \text{Sin}[x] C_2\}\}$

$\mathbf{DSolve}[\{\boldsymbol{\partial}_{\boldsymbol{xx}}\boldsymbol{y}[\boldsymbol{x}] + \boldsymbol{y}[\boldsymbol{x}] == \mathbf{0}, \boldsymbol{y}[\boldsymbol{\pi}/\mathbf{2}] == \mathbf{1}, \boldsymbol{y}'[\boldsymbol{\pi}/\mathbf{2}] == \mathbf{0}\}, \boldsymbol{y}[\boldsymbol{x}], \boldsymbol{x}]$

$\{\{y[x] \to \text{Sin}[x]\}\}$

Чтобы набрать выражение $\partial_{xx} y[x]$ вы можете ввести комбинацию $Esc - pd - Esc$, которая создает символ ∂, затем комбинацию $Ctrl - \_$ (подчеркивание), затем *x*, затем комбинацию $Esc - запятая - Esc$, которая ставит невидимую запятую, затем снова *x*, после чего появится ∂_{xx}, затем → (клавиша управления текстовым курсором), затем вводите *y*[*x*].

`DSolve` может решать линейные обыкновенные дифференциальные уравнения с постоянными коэффициентами любого порядка. Функция может решать много линейных уравнений с непостоянными коэффициентами, а также может решать много нелинейных ОДУ, методы решения которых изложены в стандартных руководствах.

Вот несколько примеров. Общее решение уравнения четвертого порядка включает 4 неопределенные константы.

$$\mathbf{DSolve}[y''''[x] == y[x], y[x], x]$$

$$\{\{y[x] \to e^{x} C[1] + e^{-x} C[3] + C[2]\text{Cos}[x] + C[4]\text{Sin}[x]\}\}$$

Каждое независимое начальное условие уменьшает на единицу количество независимых констант.

$$\mathbf{DSolve}[\{y''''[x] == y[x], y[0] == y'[0] == 0\}, y[x], x]$$

$$\{\{y[x] \to e^{-x}(-C[1] + e^{2x} C[1] - C[2] + e^{x} C[2]\text{Cos}[x] - 2e^{x} C[1]\text{Sin}[x] - e^{x} C[2]\text{Sin}[x])\}\}$$

`DSolve` умеет решать краевые задачи

$$\mathbf{DSolve}[\{y''[x] + y[x] == 0, y[0] == 1, y[\pi/2] == 0\}, y[x], x]$$

$$\{\{y[x] \to \text{Cos}[x]\}\}$$

Однако следует помнить, что не все краевые задачи имеют решение, например

$$\mathbf{DSolve}[\{y''[x] + y[x] == 0, y[0] == 1, y[\pi] == 0\}, y[x], x]$$

```
DSolve::bvnul: For some branches of the general solution, the
given boundary conditions lead to an empty solution
```

$$\{\}$$

или, наоборот, могут иметь не единственное решение

$$\mathbf{DSolve}[\{y'' [x] + y[x] == 0, y[0] == 0, y[\pi] == 0\}, y[x], x]$$

```
DSolve::bvsing: Unable to resolve some of the arbitrary constants
in the general solution using the given boundary conditions...
```

$$\{\{y[x] \to C[2]\text{Sin}[x]\}\}$$

В граничных условиях можно задавать линейную комбинацию значений функции и ее производных.

$$\mathbf{DSolve}[\{y''[x] + y[x] == 0, y[0] == 1, y[\pi] + y'[\pi] == 0\}, y[x], x]$$

$$\{\{y[x] \to \text{Cos}[x] - \text{Sin}[x]\}\}$$

Определение точного решения ОДУ является сложной задачей, и далеко не все уравнения имеют общее решение в символьном виде. Легче всего решаются линейные ОДУ

$$\mathbf{DSolve}[\{xy'[x] == 3y[x] + x^4\mathbf{Cos}[x], y[2\pi] == 0\}, y[x], x]$$

$$\{\{y[x] \to x^3\text{Sin}[x]\}\}$$

$$\mathbf{DSolve}[y'[x] - x\, y[x] == 0, y[x], x]$$

$$\{\{y[x] \to e^{\frac{x^2}{2}} C[1]\}\}$$

Добавление неоднородности усложняет вид решения.

$$\mathbf{DSolve}[y'[x] - x\, y[x] == 1, y[x], x]$$

$$\{\{y[x] \to e^{\frac{x^2}{2}} C[1] + e^{\frac{x^2}{2}} \sqrt{\frac{\pi}{2}}\,\text{Erf}[\frac{x}{\sqrt{2}}]\}\}$$

Здесь `Erf[x]` – функция (интеграл) ошибок. Вот пример нелинейного ОДУ

$$\mathbf{DSolve}[x^2 y''[x] - y'[x]^2 == 0, y[x], x]$$

$$\{\{y[x] \to -\frac{x}{C[1]} + C[2] - \frac{\text{Log}[1 - xC[1]]}{C[1]^2}\}\}$$

Решение многих ДУ может быть представлено только через специальные функции. В следующем уравнении решение определяется через функции Эйри

$$\mathbf{DSolve}[y''[x] - x\, y[x] == 0, y[x], x]$$

$$\{\{y[x] \to \text{AiryAi}[x]C[1] + \text{AiryBi}[x]C[2]\}\}$$

Решение следующего уравнения выражается через функции Бесселя.

$$\mathbf{DSolve[x^2 y''[x] + x\, y'[x] + (x^2 - n^2) y[x] == 0, y[x], x]}$$

$$\{\{y[x] \to \text{BesselJ}[n, x] C[1] + \text{BesselY}[n, x] C[2]\}\}$$

Решения, возвращаемые функцией `DSolve`, иногда включает интегралы, которые не могут быть точно вычислены в символьном виде. Также иногда `DSolve` дает представление решения в неявной форме с использованием функции `Solve`. Следующее уравнение Абеля имеет представление решения в виде неявно определяемой функции.

$$\mathbf{DSolve}[y'[x] + xy[x]^3 + y[x]^2 == 0, y[x], x]$$

$$\text{Solve}[\frac{1}{2}(\frac{2\text{ArcTanh}[\frac{-1-2xy[x]}{\sqrt{5}}]}{\sqrt{5}} + \text{Log}[\frac{-1-xy[x](-1-xy[x])}{x^2 y[x]^2}]) == C[1] - \text{Log}[x], y[x]]$$

Есть возможность создавать собственное обозначение для производных, которые можно использовать при записи ДУ.

$\mathbf{dx} := \boldsymbol{D}[\#, x]\&$ (пользовательское обозначение функции дифференцирования)

$\mathbf{dx}[f[x]]$

$f'[x]$

или

$\mathbf{dx} = \mathbf{Function}[\{f\}, \boldsymbol{D}[f, x]]$

$\mathbf{dx@}f[x]$

$f'[x]$

Здесь символ @ обозначает префиксное использование/применение функции пользователя `dx` к последующему аргументу `f[x]`. В новых обозначениях например, запись

$\mathbf{dx@}y[x] == -x^3 y[x]$

будет представлять дифференциальное уравнение

$y'[x] == -x^3 y[x]$

Допустимо использование полной формы функции дифференцирования, например

$\mathbf{Derivative}[2][y][x]$

$y''[x]$

После решения ОДУ можно выполнить проверку полученного решения с помощью подстановки

$$\mathbf{ds} = \mathbf{DSolve}[\{y''[x] - y[x] == 0, y[0] == 1, y'[0] == 4\}, y, x]$$

$$\{\{y \to \text{Function}[\{x\}, \frac{1}{2} e^{-x}(-3 + 5e^{2x})]\}\}$$

$$\{y''[x] - y[x] == 0, y[0] == 1, y'[0] == 4\}/.\, \mathbf{ds}$$

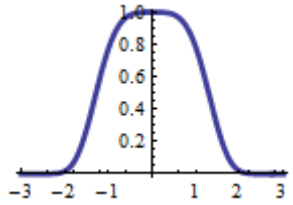
```
{{True, True, True}}
```

Если мы хотим построить график решения, то список правил подстановки надо преобразовать в выражение, которое следует передать функции `Plot`. Это делается следующим образом

$$\mathbf{ds} = \mathbf{DSolve}[\{y'[x] == -x^3 y[x], y[0] == 1\}, y[x], x]$$

$$\{\{y[x] \to e^{-\frac{x^4}{4}}\}\}$$

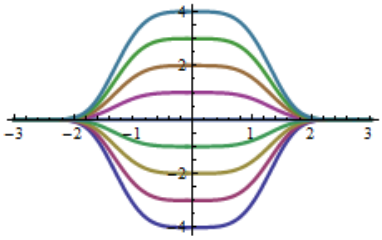
$$\mathbf{Plot}[y[x]/.\, \mathbf{ds}, \{x, -3, 3\}, \mathbf{PlotStyle} -> \mathbf{Thickness}[0.01]]$$



Можно построить график решений при разлчных значениях постоянной, вместо того, чтобы использовать параметр в начальных условиях.

$$\mathbf{ds = DSolve[\{y'[x] == -x^3 y[x]\}, y[x], x];}$$
$$\mathbf{tab = Table[y[x]/.ds[[1]]/.\{C[1] \to k\}, \{k, -4, 4, 1\}]}$$
$$\mathbf{Plot[Evaluate[tab], \{x, -3, 3\}, PlotStyle \to \{Thickness[0.01]\}]}$$

$$\{-4e^{-\frac{x^4}{4}}, -3e^{-\frac{x^4}{4}}, -2e^{-\frac{x^4}{4}}, -e^{-\frac{x^4}{4}}, 0, e^{-\frac{x^4}{4}}, 2e^{-\frac{x^4}{4}}, 3e^{-\frac{x^4}{4}}, 4e^{-\frac{x^4}{4}}\}$$



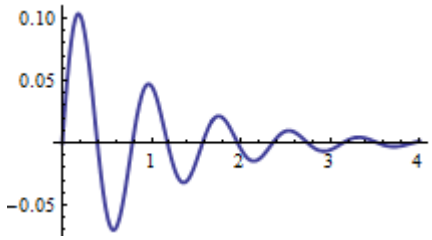
Часто нам требуется решение в виде функции или выражения, которые мы собираемся использовать не только для построения графика. Например, при построении фазовых кривых ДУ 2 – го порядка нам нужна еще и производная решения. Тогда список правил подстановки можно преобразовать в функцию.

$$\mathbf{ds = DSolve[\{y''[x] + 2y'[x] + 65y[x] == 0, y[0] == 0, y'[0] == 1\}, y, x]}$$

$$\{\{y \to \text{Function}[\{x\}, \frac{1}{8} e^{-x} \text{Sin}[8x]]\}\}$$

Для того, чтобы последний список списков преобразовать в одинарный список надо избавиться от одной пары охватывающих фигурных скобок, например, взять `ds[[1]]`. Можно избавиться от обеих пар охватывающих скобок, взяв `ds[[1,1]]`. Подставив этот результат вместо `y` с помощью операции подстановки **/.** (слэш и точка), получим функцию, представляющую решение ОДУ

$$\mathbf{z = y/.ds[[1, 1]]}$$
$$\mathbf{Plot[z[x], \{x, 0, 4\}, PlotRange \to All]}$$



Созданную функцию `z` вы можете использовать для дифференцирования, интегрирования, вычисления значения в точке и в любых других ситуациях, в которых уместно использовать функцию.

Ту же функцию можно создать следующим образом

$$\mathbf{z[x\_] := (y/.ds[[1]])[x]}$$

Мы создали функцию `z[x]`, представляющую решение, используя подстановку `{y→Function[...]}`.

Решение можно искать в виде выражения `y[x]`.

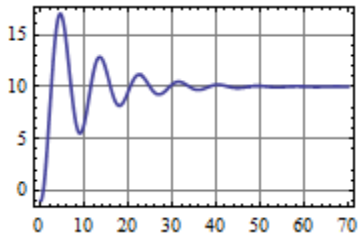
$$\mathbf{ds = DSolve[\{4y''[x] + 4/5y'[x] + 2y[x] == 20, y[0] == -1, y'[0] == 0\}, y[x], x]}$$

$$\{\{y[x] \to \frac{1}{7} e^{-x/10}(70e^{x/10} - 77\text{Cos}[\frac{7x}{10}] - 11\text{Sin}[\frac{7x}{10}])\}\}$$

Тогда для построения функции, представляющей решение, можно поступить следующим образом

$$\mathbf{z[t\_] := (y[x]/.ds[[1]])/.x \to t}$$

$$\mathbf{Plot[z[x], \{x, 0, 70\}, PlotRange \to All, PlotStyle \to Thickness[0.01], GridLines \to Automatic, Frame \to True]}$$



Не забывайте удалять значения переменных из рабочего пространства. Так перед предыдущим кодом следует выполнить команду `Remove[z]`, чтобы «забыть» определение функции `z`, сделанное в предыдущем примере.

Функцию решения можно создать еще так

$$\mathbf{z[x\_] = y[x]/.ds[[1, 1]]}$$

$$\mathbf{Plot[z[x], \{x, 0, 70\}, PlotRange -> } All]$$

Построить график можно и следующими командами

$$\mathbf{z[x\_] := y[x]/.ds[[1]]}$$ (* отложенное присваивание *)

$$\mathbf{Plot[Evaluate[z[x]], \{x, 0, 70\}, PlotRange -> } All]$$

Однако «настоящей» функции команда `z[x_]:=y[x]/.ds[[1]]` не создает, т.к. значение функции в точке, например, `z[u]`, или производная функции `z'[x]` вычисляться не будут. При построении графика команда `Evaluate[z[x]]` заставляет ядро *Mathematica* сначала вычислить `z[x]` (создать и загрузить выржение `z[x]` в рабочее пространство) и только потом функция `Plot` использует ее для построения графика.

Есть еще один способ построения функции, представляющей решение ДУ. Для этого надо выделить *выражение* из правой части подстановки $y[x] -> выражение$, которое затем использовать при создании функции

$$\mathbf{ds = DSolve[\{y''[x] + 2x\, y'[x] == 0, y[0] == 0, y'[0] == 1\}, y[x], x]}$$

$$\{\{y[x] \to \frac{1}{2}\sqrt{\pi}\text{Erf}[x]\}\}$$

Полная форма последнего результата может быть получена следующей командой

$$\mathbf{FullForm[ds[[1, 1]]]}$$

$$\text{Rule}[y[x], \text{Times}[\text{Rational}[1,2], \text{Power}[\text{Pi}, \text{Rational}[1,2]], \text{Erf}[x]]]$$

Замена имени функции «`Rule`» на имя функции «`List`» вернет результат в виде списка, составленного из левой и правой частей подстановки

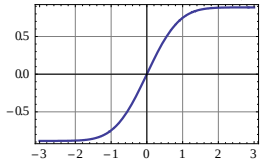
$\{l, r\} = \mathbf{ds}[[1,1]]/.\{\mathbf{Rule} \to \mathbf{List}\}$

$\{y[x], \frac{1}{2}\sqrt{\pi}\mathrm{Erf}[x]\}$

Второй элемент **r** списка содержит формульное выражение решения и его можно использовать для построения функции

$\boldsymbol{z}[\mathrm{t\_}] := \boldsymbol{r}/.\boldsymbol{x} \to \boldsymbol{t}$

$\mathbf{Plot}[\boldsymbol{z}[\boldsymbol{x}], \{\boldsymbol{x}, -\mathbf{3}, \mathbf{3}\}, \mathbf{GridLines} -> \mathbf{Automatic}, \mathbf{Frame} -> \mathbf{True}]$



Тот же график можно построить следующим образом

$\mathbf{Plot}[\boldsymbol{r}, \{\boldsymbol{x}, -\mathbf{3}, \mathbf{3}\}, \mathbf{GridLines} -> \mathbf{Automatic}, \mathbf{Frame} -> \mathbf{True}]$

Кроме графиков решений, полезны графики фазовых траекторий. Для ДУ $2-$го порядка они строятся на плоскости (y, y'). Вот как можно построить фазовую траекторию, используя созданную функцию решения.

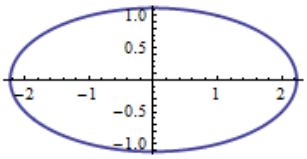
$\mathbf{ds} = \mathbf{DSolve}[\{\boldsymbol{y}''[\boldsymbol{x}] + \frac{\mathbf{1}}{\mathbf{4}}\boldsymbol{y}[\boldsymbol{x}] == \mathbf{0}, \boldsymbol{y}[\mathbf{0}] == \mathbf{1}, \boldsymbol{y}'[\mathbf{0}] == \mathbf{1}\}, \boldsymbol{y}, \boldsymbol{x}]$

$\{\{y \to \mathrm{Function}[\{x\}, \mathrm{Cos}[\frac{x}{2}] + 2\mathrm{Sin}[\frac{x}{2}]]\}\}$

$\boldsymbol{X}[\mathrm{t\_}] := (\boldsymbol{y}/.\mathbf{ds}[[\mathbf{1}]])[\boldsymbol{t}]$

$\boldsymbol{Y}[\mathrm{t\_}] := \boldsymbol{X}'[\boldsymbol{t}]$

$\mathbf{ParametricPlot}[\{\boldsymbol{X}[\boldsymbol{t}], \boldsymbol{Y}[\boldsymbol{t}]\}, \{\boldsymbol{t}, \mathbf{0}, \mathbf{4}\boldsymbol{\pi}\}]$



Покажем еще один способ выделения правой части подстановки вида $y[x] -> выражение$. Для этого вспомним, что к элементам выражений мы можем обращаться как к элементам списков. Например

$\boldsymbol{z} = \boldsymbol{f}[\boldsymbol{g}[\boldsymbol{a}], \boldsymbol{h}[\boldsymbol{b}]]$

$\{\boldsymbol{z}[[\mathbf{0}]], \boldsymbol{z}[[\mathbf{1}]], \boldsymbol{z}[[\mathbf{2}]], \boldsymbol{z}[[\mathbf{2},\mathbf{0}]], \boldsymbol{z}[[\mathbf{2},\mathbf{1}]]\}$

$\{f, g[a], h[b], h, b\}$

Здесь для выделения элементов выражения `z` мы используем индексацию. Так `z[[0]]=f` – имя внешней функции; `z[[1]]=g[a]` – первый аргумент функции, `z[[2]]=h[b]` – второй аргумент; `z[[2,0]]=h` – имя функции, стоящей во втором аргументе; `z[[2,1]]=b` – аргумент второй функции. Поскольку запись $y[x] -> выражение$ на самом деле имеет форму $Rule[y[x], выражение]$ (внешняя функция `Rule`), то для выделения правой части подстановки можно использовать подходящую индексацию. Например,

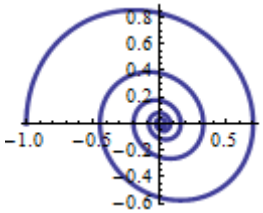
$\mathbf{ds} = \mathbf{DSolve}[\{\boldsymbol{y}'[\boldsymbol{x}] == \boldsymbol{a}\boldsymbol{y}[\boldsymbol{x}], \boldsymbol{y}[\boldsymbol{c}] == \boldsymbol{b}\}, \boldsymbol{y}[\boldsymbol{x}], \boldsymbol{x}]$

$\{\{y[x] \to b e^{-ac+ax}\}\}$

Тогда

$\mathbf{ds}[[\mathbf{1},\mathbf{1}]]$

$y[x] \to b e^{-ac+ax}$
и
$\mathbf{ds[[1,1]][[1]]}$
$y[x]$
Наконец выделяем правую часть из подстановки
$\mathbf{ds[[1,1]][[2]]}$
$b e^{-ac+ax}$
Решим ДУ и построим его фазовую траекторию
$\mathbf{ds = DSolve[\{64y''[x] + 16y'[x] + 65y[x] == 1, y[0] == -1, y'[0] == 0\}, y[x], x]}$
$\{\{y[x] \to \frac{1}{260} e^{-x/8}(4e^{x/8} - 264\text{Cos}[x] - 33\text{Sin}[x])\}\}$
$\mathbf{X[x\_] = ds[[1,1]][[2]]}$
$\mathbf{Y[x\_] := X'[x]}$
$\mathbf{ParametricPlot[\{X[t], Y[t]\}, \{t, 0, 30\}, PlotStyle -> Thickness[0.01]]}$



Для выделения выражения, стоящего в правой части подстановки `ds`, мы использовали индексацию `ds[[1,1]][[2]]`. Конечно допустимо использовать `ds[[1,1,2]]`. Обратите также внимание на присваивание при создании функции `X[x]`. Если при создании функции использовать отложенное присваивание «`:=`», то код работать не будет, т.к. функция должна быть вычислена и загружена в рабочее пространство до дифференцирования в следующей строке.

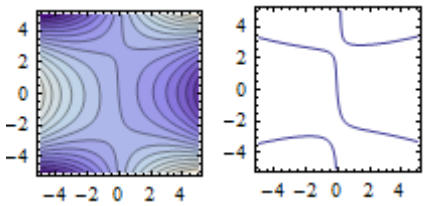
Если решение получено в неявном виде, то графическая интерпретация тоже может быть полезна.

$$\mathbf{s = DSolve}[y'[x] == \frac{x^2 - 2y[x]^2 + 14}{\text{Sin}[y[x]] + 4xy[x] - 1}, y[x], x]$$

$$\text{Solve}[-\frac{x^3}{3} - \text{Cos}[y[x]] - y[x] + 2x(-7 + y[x]^2) == C[1], y[x]]$$

Решение ОДУ было сведено к решению алгебраического уравнения. Можно построить контурный график левой части алгебраического уравнения или только «нулевые» линии контура

$\mathbf{sn = s[[1,1]]/.\{y[x] \to y\}}$
$\mathbf{ContourPlot[sn, \{x, -5, 5\}, \{y, -5, 5\}, Contours \to 15]}$
$\mathbf{ContourPlot[sn == 0, \{x, -5, 5\}, \{y, -5, 5\}]}$



Иногда решение ОДУ возвращается с использованием объекта `InverseFunction` (обратная функция).

$$\mathbf{sol} = \mathbf{DSolve}[\boldsymbol{y}''[\boldsymbol{x}] + \boldsymbol{y}[\boldsymbol{x}]\boldsymbol{y}'[\boldsymbol{x}]^{\mathbf{4}} == \mathbf{0}, \boldsymbol{y}, \boldsymbol{x}]$$

$$\{\{y \to \text{Function}[\{x\}, \text{InverseFunction}[C[1]\text{Log}[\#1 + \sqrt{-2C[1] + \#1^2}] - \frac{1}{2}\#1\sqrt{-2C[1] + \#1^2}\&][x + C[2]]]\},$$

$$\{\ y \to \text{Function}[\{x\}, \text{InverseFunction}[-C[1]\text{Log}[\#1 + \sqrt{-2C[1] + \#1^2}] + \frac{1}{2}\#1\sqrt{-2C[1] + \#1^2}\&][x + C[2]]]\}\}$$

Чтобы записать его в боле привычном виде, выполним команды

$$\mathbf{soly} = \mathbf{sol}\big[[\mathbf{1}, \mathbf{1}, \mathbf{2}, \mathbf{2}]\big]$$

$$\text{InverseFunction}[C[1]\text{Log}[\#1 + \sqrt{-2C[1] + \#1^2}] - \frac{1}{2}\#1\sqrt{-2C[1] + \#1^2}\&][x + C[2]]$$

Чтобы понять, что делает последняя команда, мы не стали завершать ее точкой с запятой. Она выделила выражение обратной функции (присутствует аргумент функции) из правила подстановки $y \to \ldots$. Для понимания происходящего выполним команду

$$\mathbf{Head}[\mathbf{soly}]$$

$$\text{InverseFunction}[C[1]\text{Log}[\#1 + \sqrt{-2C[1] + \#1^2}] - \frac{1}{2}\#1\sqrt{-2C[1] + \#1^2}\&]$$

Эта команда выделяет «чистую» обратную функцию (нет именованного аргумента). Наконец команда

$$\mathbf{Head}[\mathbf{soly}][[\mathbf{1}]][\boldsymbol{y}[\boldsymbol{x}]] == \mathbf{soly}[[\mathbf{1}]]$$

$$C[1]\text{Log}[y[x] + \sqrt{-2C[1] + y[x]^2}] - \frac{1}{2}y[x]\sqrt{-2C[1] + y[x]^2} == x + C[2]$$

выделяет выражение неявного уравнения. Фактически, целью преобразований было выделение аргумента функции `InverseFunction` командой `Head[soly][[1]]`.

Последнее выражение представляет неявное выражение функции $y(x)$ через x и две произвольные постоянные. Полученное уравнение можно представить графически с использованием контурного графика (при некоторых значениях постоянных).

$$\mathbf{eq} = \mathbf{Head}[\mathbf{soly}][[\mathbf{1}]][\boldsymbol{y}] == \mathbf{soly}[[\mathbf{1}]]/.\{\boldsymbol{C}[\mathbf{1}] \to \mathbf{1}, \boldsymbol{C}[\mathbf{2}] \to \mathbf{1}\};$$

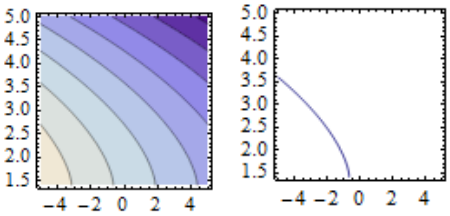
$$\boldsymbol{z} = \mathbf{eq}[[\mathbf{1}]] - \mathbf{eq}[[\mathbf{2}]]$$

$$-1 - x - \frac{1}{2}y\sqrt{-2 + y^2} + \text{Log}[y + \sqrt{-2 + y^2}]$$

В последней строке записана левая часть неявного уравнения. Теперь для этого выражения можно строить контурные графики.

$$\mathbf{ContourPlot}[\boldsymbol{z}, \{\boldsymbol{x}, -\mathbf{5}, \mathbf{5}\}, \{\boldsymbol{y}, \sqrt{\mathbf{2}}, \mathbf{5}\}]$$

$$\mathbf{ContourPlot}[\boldsymbol{z} == \mathbf{0}, \{\boldsymbol{x}, -\mathbf{5}, \mathbf{5}\}, \{\boldsymbol{y}, \sqrt{\mathbf{2}}, \mathbf{5}\}]$$



Правый график представляет график решения.

Для построения нескольких графиков можно использовать функцию `Table`. Она создает списки, элементами которых могут быть объекты различной природы, в частности это могут быть графические объекты (графики).

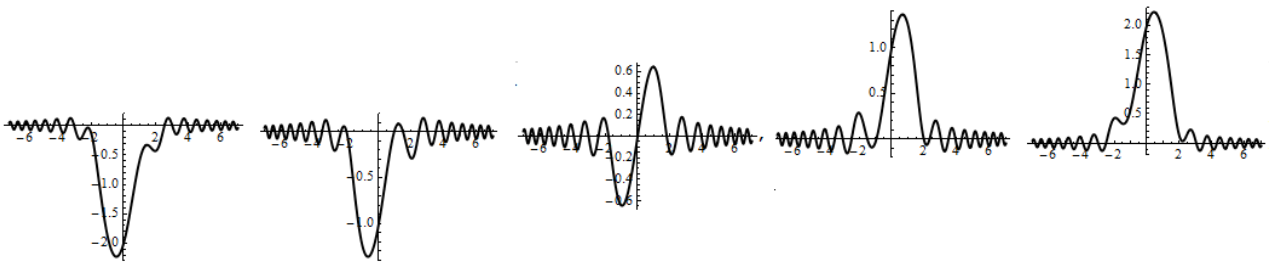
$$\mathbf{sol} = \mathbf{DSolve}[\boldsymbol{y}'[\boldsymbol{x}] + \boldsymbol{x}\boldsymbol{y}[\boldsymbol{x}] == \mathbf{Cos}[\boldsymbol{x}^2], \boldsymbol{y}, \boldsymbol{x}]$$

$$\mathbf{ts} = \mathbf{Table}[\mathbf{sol}/.\boldsymbol{C}[\mathbf{1}] \to \boldsymbol{i}, \{\boldsymbol{i}, -\mathbf{4}, \mathbf{4}, \mathbf{2}\}];$$

$$\mathbf{Table}[\mathbf{Plot}[\mathbf{Evaluate}[\boldsymbol{y}[\boldsymbol{x}]/.\mathbf{ts}[[\boldsymbol{i}]]], \{\boldsymbol{x}, -\mathbf{7}, \mathbf{7}\}], \{\boldsymbol{i}, \mathbf{5}\}]$$

$$\{\{y \to \text{Function}[\{x\}, e^{-\frac{x^2}{2}} C[1] + \frac{1}{2} e^{-\frac{x^2}{2}} \sqrt{\frac{\pi}{10}} (\sqrt{1+2i}\,\text{Erfi}[\sqrt{\frac{1}{2} - ix}]$$

$$+ \sqrt{1-2i}\,\text{Erfi}[\sqrt{\frac{1}{2} + ix}])]\}\}$$



Обратите внимание, что в выражении решения присутствует мнимая единица таким образом, что решение все равно остается вещественным.

Другие функции, связанные с поиском решений ОДУ.

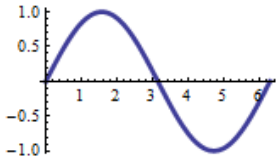
Функция `DifferentialRoot` представляет *функцию*, которая является решением линейного ОДУ.

Вот пример ДУ, решением которого является `Sin[x]`.

$$\boldsymbol{f} = \mathbf{DifferentialRoot}[\mathbf{Function}[\{\boldsymbol{y}, \boldsymbol{x}\}, \{\boldsymbol{y}''[\boldsymbol{x}] + \boldsymbol{y}[\boldsymbol{x}] == \mathbf{0}, \boldsymbol{y}[\mathbf{0}] == \mathbf{0}, \boldsymbol{y}'[\mathbf{0}] == \mathbf{1}\}]]$$

$$\text{DifferentialRoot}[\text{Function}[\{y, x\}, \{y''[x] + y[x] == 0, y[0] == 0, y'[0] == 1\}]]$$

$$\mathbf{Plot}[\boldsymbol{f}[\boldsymbol{x}], \{\boldsymbol{x}, \mathbf{0}, \mathbf{2Pi}\}]$$



`DifferentialRoot` – работает как функция

$$\boldsymbol{f}[\{\frac{\boldsymbol{\pi}}{\mathbf{6}}, \frac{\boldsymbol{\pi}}{\mathbf{4}}, \frac{\boldsymbol{\pi}}{\mathbf{3}}, \frac{\boldsymbol{\pi}}{\mathbf{2}}\}]$$

$$\{\frac{1}{2}, \frac{1}{\sqrt{2}}, \frac{\sqrt{3}}{2}, 1\}$$

$\mathbf{DifferentialRoot}[\mathbf{Function}[\{y,x\},\{y''[x]+y[x]==0,y[0]==0,y'[0]==1\}]][1]$
Sin[1]
$f[x]//\mathbf{FunctionExpand}$
Sin[x]
Функция `FunctionExpand`, использованная здесь, умеет упрощать выражения, содержащие специальные, тригонометрические и некоторые другие функции (такие как `DifferentialRoot`).

Функция `DifferentialRootReduce[expr,x]` пытается привести выражение `expr` к виду `DifferentialRoot` как функции `x`. Используя функцию `DifferentialRootReduce` можно получить задачу (дифференциальное уравнение и начальные условия), которому удовлетворяет функция

$\mathbf{DifferentialRootReduce}[\mathbf{Cos}[x],x][[0,1]]$
DifferentialRoot[Function[{y,x},{y[x]+y''[x]==0,y[0]==1,y'[0]==0}]][x]
$\%[[0,1]][y,x]$
$\{y[x]+y''[x]==0,y[0]==1,y'[0]==0\}$

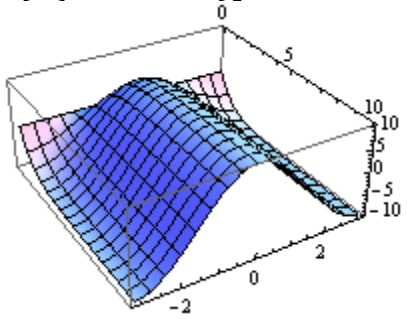
Чтобы понять, как работает последняя введенная команда мы должны вспомнить как работает индексация в выражениях. Команда `%[[0]]` вернет функцию `DifferentialRoot[Function[{y,x},...]]`, а команда `%[[0,1]]` вернет функцию `Function[{y,x},...]`. Использование команды $\%[[0,1]][y,x]$ возвращает выражение, вычисляемое функцией, т.е. $\{y[x]+y''[x]==0,y[0]==1,y'[0]==0\}$.

Вот еще примеры на использование этой функции для получения «краевой задачи»

$\mathbf{DifferentialRootReduce}[\mathbf{Exp}[-x]\mathbf{Cos}[x],x][[0,1]][y,x]$
$\{2y[x]+2y'[x]+y''[x]==0,y[0]==1,y'[0]==-1\}$
$\mathbf{DifferentialRootReduce}[\mathbf{Sqrt}[x],x][[0,1]][y,x]$
$\{-y[x]+2xy'[x]==0,y[1]==1\}$

Искомая функция может зависеть от параметра. Например

$\mathbf{sol}=\mathbf{DSolve}[\{D[x[t,\alpha],\{t,2\}]==-D[x[t,\alpha],t],$
$x[0,\alpha]==0,\mathbf{Derivative}[1,0][x][0,\alpha]==10\,\mathbf{Cos}[\alpha]\},x[t,\alpha],t]$
$\{\{x[t,\alpha]\to 10e^{-t}(-1+e^{t})\mathrm{Cos}[\alpha]\}\}$
$\mathbf{Plot3D}[x[t,\alpha]/.\mathbf{sol},\{t,0,10\},\{\alpha,-\mathbf{Pi},\mathbf{Pi}\}]$



Следует помнить, что в общем виде решения уравнения с параметром могут обнаружиться значения параметра, для которых решение недопустимо. Например

$$\mathbf{sol = DSolve[\{y'[x] == xy[x], y[0] == 1/a\}, y, x]}$$

$$\{\{y \to \text{Function}[\{x\}, \frac{e^{\frac{x^2}{2}}}{a}]\}\}$$

Очевидно, что значение a=0 недопустимо. В этом примере это видно из постановки задачи, однако могут встретиться и более сложные случаи.

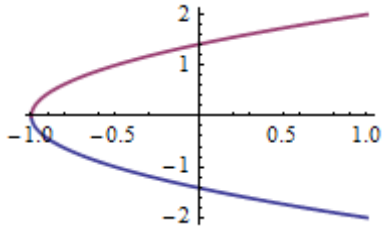
Приведем еще несколько замечаний, касающихся функции `DSolve`.

1^0. Даже для простых уравнений функция `DSolve` может найти несколько решений.

$$\mathbf{gs = DSolve[\{y'[x] == 1/y[x]\}, y, x]}$$

$$\{\{y \to \text{Function}[\{x\}, -\sqrt{2}\sqrt{x + C[1]}]\}, \{y \to \text{Function}[\{x\}, \sqrt{2}\sqrt{x + C[1]}]\}\}$$

$$\mathbf{Plot[Evaluate[y[x]/.gs/.\{C[1] \to 1\}], \{x, -1, 1\}]}$$



Задание начального условия выделяет одну ветвь решения

$$\mathbf{gs = DSolve[\{y'[x] == 1/y[x], y[0] == 1\}, y, x]}$$

```
DSolve::bvnul: For some branches of the general solution,
the given boundary conditions lead to an empty solution.
```

$$\{\{y \to \text{Function}[\{x\}, \sqrt{1 + 2x}]\}\}$$

При этом мы получаем сообщение, что могут существовать другие ветви решения (возможно решение не единственно). Если вы не желаете видеть сообщение, то используйте функцию `Quiet`.

$$\mathbf{gs = DSolve[\{y'[x] == 1/y[x], y[0] == 1\}, y, x]//Quiet}$$

$$\{\{y \to \text{Function}[\{x\}, \sqrt{1 + 2x}]\}\}$$

2^0. В системе уравнений вместо запятой можно использовать символы && (логическое И). Так следующие две команды дают одинаковый результат.

$$\mathbf{DSolve[\{y'[x] == y[x]\ \&\&\ y[0] == 1\}, y, x]}$$

$$\mathbf{DSolve[\{y'[x] == y[x], y[0] == 1\}, y, x]}$$

$$\{\{y \to \text{Function}[\{x\}, e^x]\}\}$$

3^0. Иногда, когда функция `DSolve` не дает результата, можно попробовать решить задачу относительно обратной функции, поменяв ролями зависимую и независимую переменные.

$$\mathbf{DSolve[y'[x] == 1/(x - y[x])\&\&y[0] == 1, y, x]}$$

```
Solve::ifun: Inverse functions are being used ...
{ }
```

$$\mathbf{DSolve[x'[y] == (x[y] - y)\&\&x[1] == 0, x, y]}$$

$$\{\{x \to \text{Function}[\{y\}, \frac{e - 2e^{y} + ey}{e}]\}\}$$

Во втором случае мы записали задачу относительно функции *x*(*y*).

4.1.2 Символьное решение систем ОДУ

Функция `DSolve[{eqn1, eqn2,… }, {y1, y2,…}, x]` умеет решать системы дифференциальных уравнений, которые при записи необходимо объединить в список. В таком случае в качестве второго аргумента `Dsolve` надо задавать список имен искомых функций.

$$\mathbf{DSolve}[\{y'[x] == -z[x], z'[x] == -y[x]\}, \{y[x], z[x]\}, x]$$

$$\{\{y[x] \to \frac{1}{2}e^{-x}(1 + e^{2x})C[1] - \frac{1}{2}e^{-x}(-1 + e^{2x})C[2],$$
$$z[x] \to -\frac{1}{2}e^{-x}(-1 + e^{2x})C[1] + \frac{1}{2}e^{-x}(1 + e^{2x})C[2]\}\}$$

Здесь в списке правил подстановок используются выражения. Но решение той же системы можно получить с использованием функций

$$\mathbf{DSolve}[\{y'[x] == -z[x], z'[x] == -y[x]\}, \{y, z\}, x]$$

$$\{\{y \to \text{Function}[\{x\}, \frac{1}{2}e^{-x}(1 + e^{2x})C[1] - \frac{1}{2}e^{-x}(-1 + e^{2x})C[2]],$$
$$z \to \text{Function}[\{x\}, -\frac{1}{2}e^{-x}(-1 + e^{2x})C[1] + \frac{1}{2}e^{-x}(1 + e^{2x})C[2]]\}\}$$

Заметим, что в обоих случаях решение включает две константы. Та же система с начальными условиями даст решение без произвольных постоянных

$$\mathbf{DSolve}[\{y'[x] == -z[x], z'[x] == -y[x], y[0] == 0, z[0] == 1\}, \{y[x], z[x]\}, x]$$

$$\{\{y[x] \to -\frac{1}{2}e^{-x}(-1 + e^{2x}), z[x] \to \frac{1}{2}e^{-x}(1 + e^{2x})\}\}$$

Для того, чтобы полученные правила подстановки преобразовать в функции надо избавиться от внешнего списка, а из внутреннего взять первое или второе правило для подстановки соответственно в функции x и y. Например

$$\mathbf{ds} = \mathbf{DSolve}[\{x'[t] == y[t], y'[t] == x[t], x[0] == 0, y[0] == 1\}, \{x, y\}, t]$$

$$\{\{x \to \text{Function}[\{t\}, \frac{1}{2}e^{-t}(-1 + e^{2t})], y \to \text{Function}[\{t\}, \frac{1}{2}e^{-t}(1 + e^{2t})]\}\}$$

Выражение для представления решения мы можем получить, используя индексацию, например, так `ds[[1,1,2]][t]`. Следующая цепочка команд и их результатов поясняет это.

$$\mathbf{ds}[[\mathbf{1}, \mathbf{1}]]$$

$$x \to \text{Function}[\{t\}, \frac{1}{2}e^{-t}(-1 + e^{2t})]$$

$$\mathbf{ds}[[\mathbf{1}, \mathbf{1}, \mathbf{2}]]$$

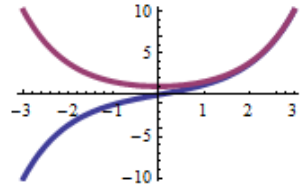
$$\text{Function}[\{t\}, \frac{1}{2}e^{-t}(-1 + e^{2t})]$$

$$\mathbf{ds}[[\mathbf{1}, \mathbf{1}, \mathbf{2}]][\boldsymbol{t}]$$

$$\frac{1}{2}e^{-t}(-1 + e^{2t})$$

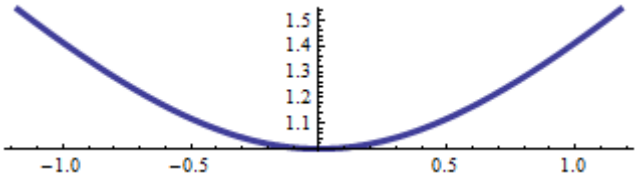
Если мы захотим построить графики функций `x[t]`, `y[t]`, представляющих решение системы ОДУ, то эти функции надо определить

$$x := \mathbf{ds}[[1,1,2]]$$
$$y := \mathbf{ds}[[1,2,2]]$$
$$\mathbf{Plot}[\{x[t], y[t]\}, \{t, -3, 3\}, \mathbf{PlotStyle} \to \mathbf{Thickness}[0.01]]$$



Фазовая траектория, соответствующая решению системы, может быть постороена с помощью функции `ParametricPlot[…]`.

$$\mathbf{ParametricPlot}[\{x[t], y[t]\}, \{t, -1, 1\}, \mathbf{PlotStyle} \to \mathbf{Thickness}[0.01]]$$

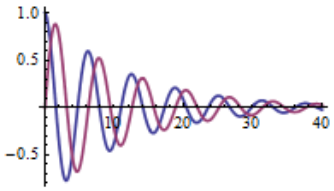


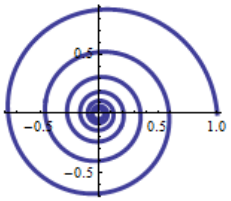
Построить фазовую траекторию можно без создания функций, представляющих решение

$$\mathbf{ds} = \mathbf{DSolve}[\{x'[t] == y[t], y'[t] == x[t], x[0] == 0, y[0] == 1\}, \{x, y\}, t]$$
$$\{\{x \to \text{Function}[\{t\}, \frac{1}{2} e^{-t}(-1 + e^{2t})], y \to \text{Function}[\{t\}, \frac{1}{2} e^{-t}(1 + e^{2t})]\}\}$$
$$\mathbf{ParametricPlot}[\{x[t], y[t]\}/.\, \mathbf{ds}, \{t, -1, 1\},$$
$$\mathbf{PlotStyle} \to \mathbf{Thickness}[0.01], \mathbf{PlotRange} \to \mathbf{All}]$$

Вот еще один пример решения системы двух ДУ первого порядка

$$\mathbf{Remove}[x, y, t]$$
$$n := 12$$
$$\mathbf{ds} = \mathbf{DSolve}[\{nx'[t] + x[t] + ny[t] == 0, ny'[t] - nx[t] + y[t] == 0,$$
$$x[0] == 1, y[0] == 0\}, \{x, y\}, t]$$
$$x = \mathbf{ds}[[1,1,2]];\ \ y = \mathbf{ds}[[1,2,2]]$$
$$\mathbf{Plot}[\{x[t], y[t]\}, \{t, 0, 40\}, \mathbf{PlotStyle} \to \mathbf{Thickness}[0.01], \mathbf{PlotRange} \to \mathbf{All}]$$
$$\mathbf{ParametricPlot}[\{x[t], y[t]\}, \{t, 0, 40\}, \mathbf{PlotStyle} \to \mathbf{Thickness}[0.02]]$$

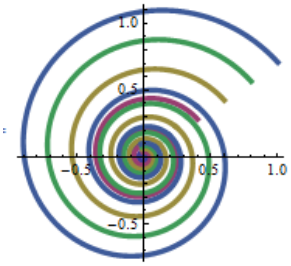




Вот как можно построить сразу несколько фазовых траекторий, соответствующих разным начальным значениям

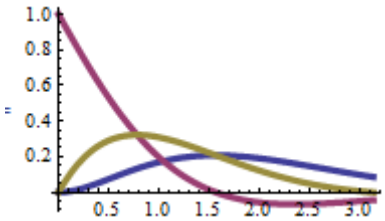
$$\mathbf{Remove}[x, y, t]$$
$$n := 8$$
$$\mathbf{ds} = \mathbf{DSolve}[\{nx'[t] + x[t] + ny[t] == 0, ny'[t] - nx[t] + y[t] == 0,$$
$$x[0] == a, y[0] == a\ \mathbf{Cos}[\frac{\pi}{4}]\}, \{x, y\}, t]$$
$$\mathbf{ParametricPlot}[\mathbf{Table}[\{x[t], y[t]\}/.\, \mathbf{ds}/.\, \{a \to m\},$$

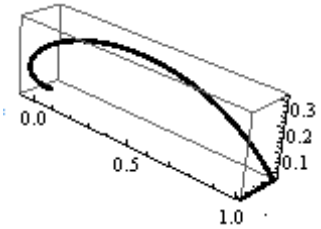
$\{m, 0.2, 1, 0.2\}], \{t, 0, 20\}$, Evaluated → True, PlotRange → All]



Вот пример построения фазовой траектории для системы 3 – х уравнений первого порядка.

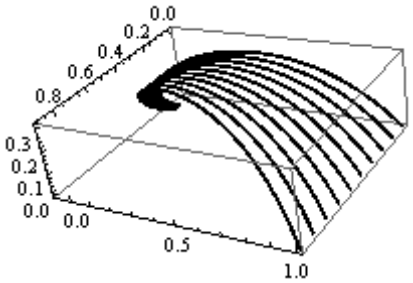
```
Remove[x, y, z, t]
ds = DSolve[{x'[t] == -x[t] + z[t], y'[t] == -y[t] - z[t],
      z'[t] == y[t] - z[t], x[0] == 0, y[0] == 1, z[0] == 0}, {x, y, z}, t]
{{x → Function[{t}, -e^-t (-1 + Cos[t])],  y → Function[{t}, e^-t Cos[t]],
  z → Function[{t}, e^-t Sin[t]]}}
x = ds[[1, 1, 2]]
y = ds[[1, 2, 2]]
z = ds[[1, 3, 2]]
Plot[{x[t], y[t], z[t]}, {t, 0, π}, PlotStyle → Thickness[0.02], PlotRange → All]
ParametricPlot3D[{x[t], y[t], z[t]}, {t, 0, π}, PlotStyle → Thickness[0.02]]
```





В следующем примере мы строим несколько фазовых траекторий той же системы ДУ, соответствующих разным начальным условиям.

```
Remove[x, y, z, fx, fy, fz, t]
sys := {x'[t] == -x[t] + z[t], y'[t] == -y[t] - z[t], z'[t] == y[t] - z[t]}
col := 10
bc := Table[{x[0] == 0.1(i - 1), y[0] == 1, z[0] == 0}, {i, 1, col}]
pp = Table[se = Join[sys, bc[[i]]];
           u = DSolve[se, {x, y, z}, t];
           fx = u[[1, 1, 2]];  fy = u[[1, 2, 2]];  fz = u[[1, 3, 2]];
           p = ParametricPlot3D[{fx[t], fy[t], fz[t]}, {t, 0, π}];
           p, {i, 1, col}];
Show[pp, PlotRange → All, AspectRatio → Automatic]
```



Систему ОДУ можно записывать, используя матрично – векторные обозначения

$\boldsymbol{A} = \{\{\mathbf{2}, -\mathbf{3}\}, \{\mathbf{1}, -\mathbf{2}\}\}$

$\boldsymbol{X}[\boldsymbol{t}\_] = \{\boldsymbol{x}[\boldsymbol{t}], \boldsymbol{y}[\boldsymbol{t}]\};$

$\mathbf{sys} = \mathbf{MapThread}[\#\mathbf{1} == \#\mathbf{2}\&, \{\boldsymbol{X}'[\boldsymbol{t}], \boldsymbol{A}.\boldsymbol{X}[\boldsymbol{t}]\}]$

$\{x'[t] == 2x[t] - 3y[t], y'[t] == x[t] - 2y[t]\}$

Теперь систему можно решить

$\mathbf{sol} = \mathbf{DSolve}[\mathbf{sys}, \{\boldsymbol{x}, \boldsymbol{y}\}, \boldsymbol{t}]$

$$\{\{x \to \text{Function}[\{t\}, \frac{1}{2}e^{-t}(-1 + 3e^{2t})C[1] - \frac{3}{2}e^{-t}(-1 + e^{2t})C[2]],$$
$$y \to \text{Function}[\{t\}, \tfrac{1}{2}e^{-t}(-1 + e^{2t})C[1] - \tfrac{1}{2}e^{-t}(-3 + e^{2t})C[2]]\}\}$$

4.2 Численные решения

4.2.1 Численное решение ОДУ

В тех случаях, когда символьное решение дифференциального уравнения не может быть найдено помогает функция `NDSolve`, которая позволяет численно решать дифференциальные уравнения. Она имеет следующий синтаксис

```
NDSolve[{уравнения}, y, {x, xmin, xmax}]
```

где `уравнения` содержит запись ОДУ и начальных/граничных условий относительно неизвестной функции `y` и независимой переменной `x`, изменяющейся в пределе от x_{min} до x_{max}. `NDSolve` дает решение в виде объекта/функции `InterpolatingFunction`. Список уравнений должен содержать достаточное количество начальных и/или граничных условий для полного определения решения. Начальные и граничные условия обычно задаются в виде $y[x_0] == c_0$, $y'[x_0] == b_0$ и т.д.

Функция `NDSolve` имеет такие же аргументы, как и `DSolve`, но с некоторыми ограничениями. Надо определить достаточно начальных/граничных условий, чтобы получить полностью определенное решение. Надо также определить интервал изменения независимой переменной (на этом интервале решение должно существовать).

$\mathbf{nds} = \mathbf{NDSolve}[\{\boldsymbol{y}'[\boldsymbol{x}] == \boldsymbol{x}^{\mathbf{2}}, \boldsymbol{y}[\mathbf{0}] == \mathbf{1}\}, \boldsymbol{y}[\boldsymbol{x}], \{\boldsymbol{x}, -\mathbf{3}, \mathbf{3}\}]$

$\{\{y[x] \to \text{InterpolatingFunction}[\{\{-3.,3.\}\}, \text{" <> "}][x]\}\}$

Результат выдается в таком же формате, что и для `Solve` или `DSolve`, за исключением того, что решение является объектом типа `InterpolatingFunction`. Это черный ящик: задаешь входное значение и получаешь выходное, но не можешь видеть ничего внутри. Обычно лучшим способом использования `InterpolatingFunction` считается построение графика или вычисление значений функции в точках. Например, создадим функцию

$\boldsymbol{z}[\mathbf{x}\_] = \mathbf{nds}[[\mathbf{1}, \mathbf{1}, \mathbf{2}]]$

$\{\boldsymbol{z}[\mathbf{0}], \boldsymbol{z}[\mathbf{1}], \boldsymbol{z}[\mathbf{3}]\}$

```
{1., 1.33333, 10.}
```

Начнем рассмотрение примеров.

$$\mathbf{nds = NDSolve[\{y'[x] == y[x], y[0] == 2\}, y, \{x, 0, 3\}]}$$

{{$y \to$ InterpolatingFunction[{{0.,3.}}," <> "]}}

Решение представлено в виде списка подстановок, основным элементом которого является объект `InterpolatingFunction`. Для превращения списка подстановок в функцию надо поступить обычным способом

$$\mathbf{z = nds[[1, 1, 2]]}$$

$$\mathbf{\{z[0], z[1], z[3]\}}$$

$$\{2., 5.43656, 40.17107\}$$

Сравните вид искомого решения и способ создания функции `z` в этом и предыдущем примерах.

Сравним решения, полученные с помощью функции `Dsolve` и `NDSolve`.

$$\mathbf{nds = NDSolve[\{y'[x] == y[x]^2, y[0] == 1\}, y, \{x, 0, 1\}]}$$

$$\mathbf{ds = DSolve[\{y'[x] == y[x]^2, y[0] == 1\}, y, x]}$$

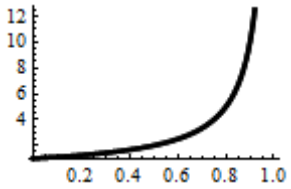
$$\mathbf{zn = nds[[1, 1, 2]]}$$

$$\mathbf{zd = ds[[1, 1, 2]]}$$

$$\mathbf{Plot[\{zn[x], zd[x]\}, \{x, 0, 1\}, PlotStyle \to \{\{Black, Thickness[0.02]\}\}]}$$

{{y→InterpolatingFunction[{{0.,1.}},<>]}}

$$\{\{y \to \text{Function}[\{x\}, \frac{1}{1-x}]\}\}$$



Графики обеих функций сливаются. Однако, обратите внимание на пределы изменения параметра `x` при численном решении. Если определить верхнюю границу изменения `x` равной 1, то будем получать сообщения о возможных ошибках, которые в данном случае означают, что точка `x=1` является особой.

Дополнительные условия можно задавать в разных точках

$$\mathbf{nds = NDSolve[\{y'''[x] + Sin[x] == 0, y[0] == 4, y[1] == 7, y[2] == 0\}, y, \{x, 0, 2\}]}$$

{{$y \to$ InterpolatingFunction[{{0.,2.}}," <> "]}}

В граничных условиях можно использовать линейные комбинации значений функции и ее производных.

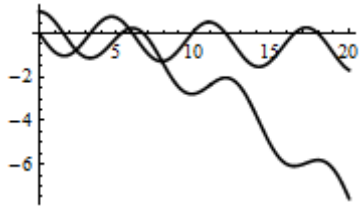
$$\mathbf{nds = NDSolve[\{y''[x] + y[x] == 12x, 2y[0] - y'[0] == -1, 2y[1] + y'[1] == 9\}, y, \{x, 0, 1\}]}$$

{{$y \to$ InterpolatingFunction[{{0.,1.}}," <> "]}}

Если все начальные условия заданы в одной точке, скажем в точке x_0, тогда для задания диапазона изменения независимой переменной можно использовать только одно значение, например, в виде `{x, x1}`. *Mathematica* сгенерирует решение в диапазоне `{x, x0, x1}`.

$$\mathbf{nds = NDSolve[\{y''[t] + y[t] == -0.02t^2, y[0] == 1, y'[0] == 0\}, y, \{t, 20\}];}$$

$$\mathbf{Plot}[\{y[t], y'[t]\}/.\,\mathbf{nds}, \{t, 0, 20\}, \mathbf{PlotStyle} \to \{\mathbf{Thickness}[0.01], \mathbf{RGBColor}[0,0,0]\}]$$



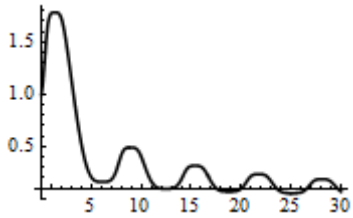
Попутно обратите внимание, что объект `InterpolatingFunction` можно дифференцировать.

Чтобы во всех примерах при построении графиков решений и фазовых траекторий постоянно не устанавливать длинный список опций, их можно установить один раз на всю рабочую сессию функцией `SetOptions`, например, так

$$\mathbf{SetOptions}[\mathbf{Plot}, \mathbf{PlotRange} \to \mathbf{All}, \mathbf{PlotStyle} \to \{\mathbf{Thickness}[0.01], \mathbf{RGBColor}[0,0,0]\}];$$

Первый аргумент `SetOptions` – имя функции для которой будут изменены опции по–умолчанию, затем следует последовательность изменяемых опций и их значений. Теперь при построении графиков функцией `Plot` мы будем строить кривые черным цветом и с увеличенной толщиной.

$$s = \mathbf{NDSolve}[\{y'[x] == \mathbf{Sin}[x + y[x]]^3 y[x], y[0] == 1\}, y, \{x, 0, 30\}]$$
$$\mathbf{Plot}[\mathbf{Evaluate}[y[x]/.s], \{x, 0, 30\}]$$

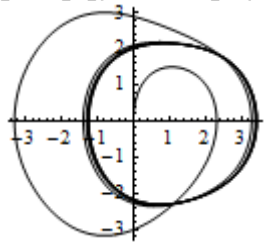


Используя функцию `SetOptions`, установим опции для функции `ParametricPlot`.

$$\mathbf{SetOptions}[\mathbf{ParametricPlot}, \mathbf{PlotRange} \to \mathbf{All}, \mathbf{PlotStyle} \to \{\mathbf{Thickness}[0.01], \mathbf{Black}\}];$$

При построения фазовой траектории можно непосредственно дифференцировать объект `InterpolatingFunction`.

$$\mathbf{nds} = \mathbf{NDSolve}[\{x''[t] + x'[t] + \mathbf{Cos}[x[t]] == 4\mathbf{Cos}[t],$$
$$x[0] == x'[0] == 0\}, x, \{t, 0, 50\}];$$
$$\mathbf{ParametricPlot}[\mathbf{Evaluate}[\{x[t], x'[t]\}/.\,\mathbf{nds}], \{t, 0, 50\}]$$



Вот пример уравнения, которое не удается решить в символьном виде

$$\mathbf{DSolve}[\{y'''[x] + y''[x] + y'[x] == -y[x]^3, y[0] == 1, y'[0] == y''[0] == 0\}, y, x]$$

DSolve[{$y'[x] + y''[x] + y^{(3)}[x] == -y[x]^3, y[0] == 1, y'[0] == y''[0] == 0\}, y, x$]

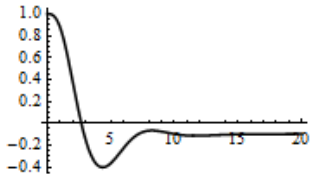
Когда *Mathematica* не может решить уравнение она возвращает исходный текст.

Численное решение этого уравнения у *Mathematica* не вызывает затруднений. Однако такое решение не представляет интереса, если его не использовать в привычном для нас виде, например в виде графика.

$$\mathbf{nds = NDSolve[\{y'''[x] + y''[x] + y'[x] == -y[x]^3, y[0] == 1,}$$
$$\mathbf{y'[0] == y''[0] == 0\}, y, \{x, 0, 20\}]}$$

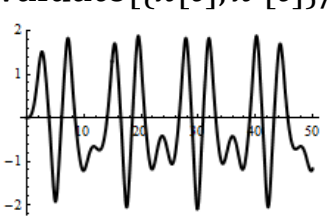
$\{\{y \to \text{InterpolatingFunction}[\{\{0.,20.\}\},\text{" <> "}]\}\}$

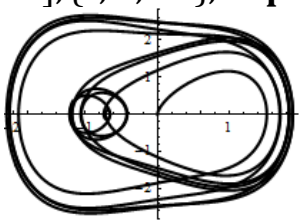
$$\mathbf{Plot[Evaluate[y[x]/.nds], \{x, 0, 20\}]}$$



Кроме графиков решений интерес представляют фазовые траектории, которые для ДУ 2-го порядка строятся на плоскости (x, x') с использованием функции `ParametricPlot`.

$$\mathbf{nds = NDSolve[\{x''[t] + x[t]^3 == Sin[t], x[0] == x'[0] == 0\}, x, \{t, 0, 50\}]}$$
$$\mathbf{Plot[x[t]/.nds, \{t, 0, 50\}]}$$
$$\mathbf{ParametricPlot[Evaluate[\{x[t], x'[t]\}/.nds], \{t, 0, 50\}, AspectRatio \to 0.7]}$$





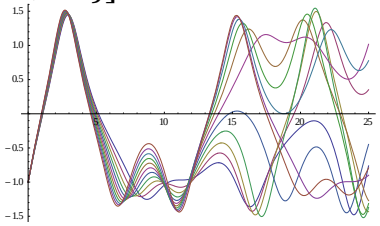
Здесь слева показан график решения, а справа – фазовая траектория. Интересно пронаблюдать в динамике движение точки по фазовой траектории.

```
Do[ParametricPlot[Evaluate[{x[t],x'[t]}/.nds],{t,0,n},
        PlotRange→{{-3,3},{-3,3}}],{n,1,50}] (в старых версиях)
```

$$\mathbf{Animate[ParametricPlot[Evaluate[\{x[t], x'[t]\}/.nds], \{t, 0, n\},}$$
$$\mathbf{PlotRange \to \{\{-3, 3\}, \{-3, 3\}\}, PlotPoints \to 1000\ ], \{n, 1, 50\}]}$$

Вот пример решения списка задач, отличающихся друг от друга начальным условием.

$$\mathbf{sol = Table[NDSolve[\{x''[t] + 0.15x'[t] - x[t] + x[t]^3 == 0.3Cos[t],}$$
$$\mathbf{x[0] == -1, x'[0] == a\}, x, \{t, 0, 25\}],}$$
$$\mathbf{\{a, 1, 1.1, 0.01\}]}$$
$$\mathbf{Plot[Evaluate[x[t]/.sol], \{t, 0, 25\}]}$$

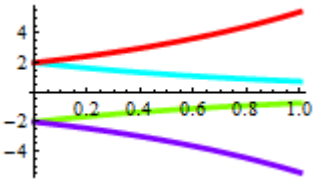


Встречаются уравнения с несколькими решениями. В следующем примере мы получаем 4 приближенных решения.

$$\mathbf{NDSolve[\{y'[x]^2 - y[x]^2 == 0, y[0]^2 == 4\}, y[x], \{x, 0, 1\}]}$$

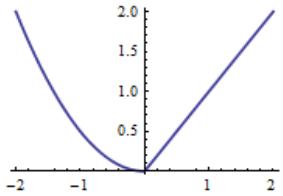
$\{\{y[x] \to \text{InterpolatingFunction}[\{\{0.,1.\}\},\text{" <> "}][x]\},$

```
 {y[x] → InterpolatingFunction[{{0.,1.}},<>][x]},
 {y[x] → InterpolatingFunction[{{0.,1.}},<>][x]},
 {y[x] → InterpolatingFunction[{{0.,1.}}," <> "][x]}}
Plot[Evaluate[y[x]/.nds],{x,0,1},PlotStyle
          → Table[{Thickness[0.02],Hue[i/4]},{i,1,4}]]
```

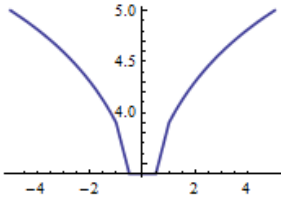


Функция `NDSolve` отлично решает кусочные ОДУ.

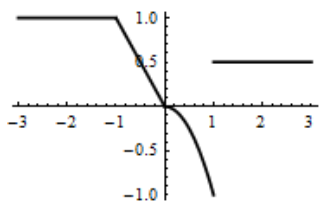
```
nds = NDSolve[{y'[x] == If[x < 0,x,1],y[0] == 0},y,{x,−2,2}]
Plot[Evaluate[y[x]/.nds],{x,−2,2},PlotStyle → Thickness[0.01]]
```

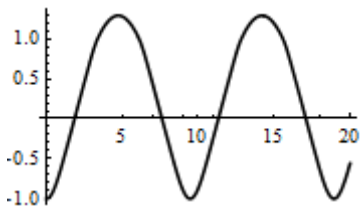


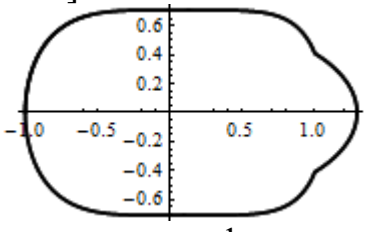

```
nds = NDSolve[{y'[x] == Which[
                    x < −1,      1/(x − 1),
                    x < −0.5,    −1,
                    x < 0.5,     0,
                    x < 1,       1,
                    x ≥ 1,       1/(x + 1)],
                y[−5] == 5},y,{x,−5,5}]
Plot[Evaluate[y[x]/.nds],{x,−5,5},PlotStyle → Thickness[0.01]]
```




```
f[x_] = Piecewise[{{1,x < −1},{−x,x < 0},{−x^2,x < 1}},0.5];
Plot[f[x],{x,−3,3}]
u = NDSolve[{x''[t] + f[x[t]]^2 x[t] == 0,x[0] == −1,x'[0] == 0},x,{t,0,20}]
Plot[x[t]/.u[[1,1]],{t,0,20}]
ParametricPlot[Evaluate[{x[t]/.u[[1,1]],x'[t]/.u[[1,1]]}],{t,0,20},
          AspectRatio → Automatic,PlotPoints → 1000]
```







На рисунке слева показана кусочная функция, в середине – график решения ОДУ, справа – фазовая траектория.

Иногда нужно остановить вычисления при достижении решением какого – либо значения или при выполнении какого – либо условия. Для этого в опциях функции `NDSolve` предусмотрено использование функции `WhenEvent`. Она имеет синтаксис

WhenEvent[*условие, действие*]

При включении этой функции в список опций `NDSolve`, на каждом шаге выполняется проверка логического выражения «*условие*» и, когда оно принимает значение `True`, происходит выполнение команды, представляющей «*действие*», при текущем значении независимой переменной. Эта команда может быть оператором подстановки, например, y[x]→*значение* или строкой `"StopIntegration"`, останавливающей вычисления. Допустимы и некоторые другие значения, с которыми вы можете познакомиться по справочной системе.

В следующем примере мы решаем ДУ до тех пор, пока решение не станет равным нулю (решением является `Cos[x]`).

$$\mathbf{nds} = \mathbf{NDSolve}[\{y''[x] + y[x] == 0, y[0] == 1, y'[0] == 0,$$
$$\mathbf{WhenEvent}[y[x] == 0, \text{"}\mathbf{StopIntegration}\text{"}]\}, y, \{x, \infty\}]$$

$$\{\{y \to \text{InterpolatingFunction}[\{\{0.,1.570796\}\},\text{" <> "}]\}\}$$

$$\mathbf{xmax} = \mathbf{nds}[[1,1,2]][[1]][[1]][[2]]$$

$$\mathbf{Plot}[y[x]/.\mathbf{nds}, \{x, 0, \mathbf{xmax}\}]$$

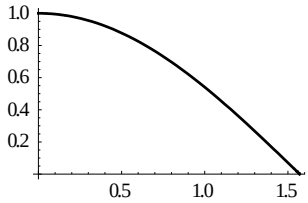
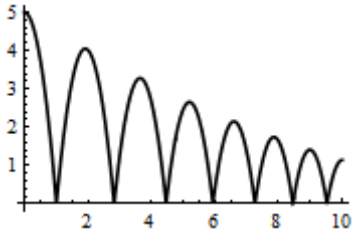


График решения построен до значения независимой переменной `x`, при котором решение обратилось в нуль. Для того, чтобы определить это значение, мы использовали индексацию.

Вот пример задачи, моделирующей движение «прыгающего» вертикально мяча (под действием силы притяжения), который теряет 10% своей скорости при отскоке от земли.

$$\mathbf{nds} = \mathbf{NDSolve}[\{y''[t] == -9.81, y[0] == 5, y'[0] == 0,$$
$$\mathbf{WhenEvent}[y[t] == 0, y'[t] -> -0.9y'[t]]\}, y, \{t, 0, 10\}];$$

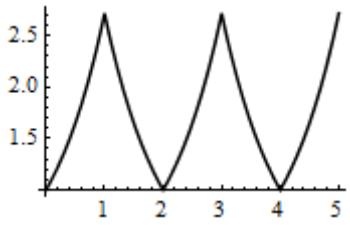
$$\mathbf{Plot}[y[t]/.\mathbf{nds}, \{t, 0, 10\}]$$



Здесь значение `y[t]` представляет вертикальную координату мяча, который начал падать с высоты 5 с нулевой начальной скоростью.

Вот пример использования этой опции для имитации кусочного ОДУ

```
sol = NDSolve[{y'[t] == a[t] y[t], y[0] == 1, a[0] == 1,
WhenEvent[Mod[t, 1] == 0, a[t] → -a[t]]}, y, {t, 0, 5}, DiscreteVariables → a[t]];
Plot[y[t] /. sol, {t, 0, 5}]
```



При численном решении уравнений важными являются следующие опции функции `NDSolve`, которые влияют на точность вычислений:

- `AccuracyGoal` определяет абсолютную погрешность вычислений на каждом шаге, используя количество значащих цифр; AccuracyGoal → ∞ говорит, что эта опция не должна использоваться как критерий прекращения вычислений (т.е. будут использоваться другие опции);
- `PrecisionGoal` определяет «относительную погрешность» на каждом шаге вычислений и задается количеством значащих цифр; PrecisionGoal → ∞ указывает, что эта опция не должна использоваться как критерий прекращения вычислений (будет использоваться опция `AccuracyGoal`);
- опция `WorkingPrecision` определяет количество значащих цифр, используемых во внутренних вычислениях; значение этой опции, как правило, должно быть больше значения опции `AccuracyGoal`; установка `WorkingPrecision->MachinePrecision` приводит к вычислениям с процессорной точностью;
- опция `MaxSteps` задает максимальное количество шагов дискретизации;
- опция `MaxStepSize` задает максимальную длину шагов дискретизации; `MaxStepSize→Infinity` отключает ограничение на размер шага;
- `StartingStepSize` задает начальный размер шага.

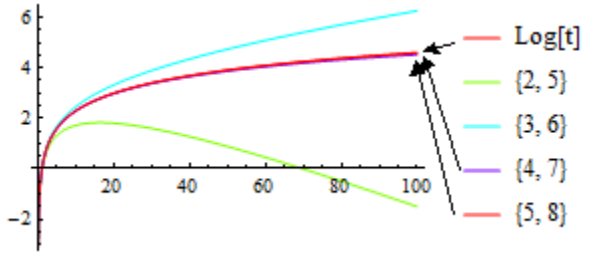
Для опции `PrecisionGoal` мы используем не совсем корректный термин «относительная погрешность». `PrecisionGoal->n` определяет для значения *x* погрешность вычислений равную $|x|10^{-n}$. Когда включены опции `AccuracyGoal->m` и `PrecisionGoal->n`, то *Mathematica* пытается выполнять вычисления величины *x* с погрешностью не более $10^{-m}+|x|10^{-n}$.

`NDSolve` подбирает размер шага так, чтобы ошибка вычислений не превосходила погрешностей, определяемых опциями `AccuracyGoal` и `PrecisionGoal`. Установка значения `Automatic` опций `AccuracyGoal` и `PrecisionGoal` эквивалентна значению `WorkingPrecision/2`.

Пример. Решим дифференциальное уравнение $y''=-\frac{1}{t^2}$ на отрезке $[a;100]$ с начальными условиями $y(a)=\ln(a)$, $y'(a)=1/a$ при a=0.001. Его точное

решение $y = \ln t$. Построим график точного решения $\ln t$ и приближенных, полученных при различных абсолютной и относительной погрешностях

```
a = 0.001;
ts = Table[
    NDSolveValue[{y''[t] == -1/t^2, y[a] == Log[a], y'[a] == 1/a}, y, {t, a, 100},
                 AccuracyGoal → p - 3, PrecisionGoal → p], {p, 5, 8}];
Plot[Evaluate[Join[{Log[t], Table[ts[[i]][t], {i, 1, 4}]}]], {t, a, 100},
      PlotStyle → Table[Hue[k], {k, 0, 1, 0.25}],
      PlotLegends → Join[{"Log[t]"}, Table[ToString[{p - 3, p}], {p, 5, 8}]]]
```



На предыдущем рисунке красной линией показан график точного решения и другими цветами графики приближенного решения при различных относительной и абсолютной погрешностях. Как видим, удовлетворительное приближение к точному решению получается при `AccuracyGoal->4` и `PrecisionGoal->7`. Меньших значений недостаточно.

Пример. Отключим `AccuracyGoal` и решим ДУ, используя только, заданную по-умолчанию, относительную погрешность вычислений

```
a = 0.000001;
si = NDSolve[{x'[t] == x[t], x[0] == a}, x, {t, 0, 1}, AccuracyGoal → ∞]
eri[t_] = 1 - (x[t]/. si)[[1]]/(a Exp[t])
eri[1]
```

-8.00399×10^{-9}

Точное решение уравнения известно $a \cdot e^{-t}$. Относительная погрешность в точке `t` определяется формулой $1 - \frac{x(t)}{a \cdot e^{-t}}$. Но нам еще пришлось выполнить замену $(x[t]/.\,\text{si})[[1]]$.

Выполним те же вычисления, не отключая опцию `AccuracyGoal` .

```
ss = NDSolve[{x'[t] == x[t], x[0] == a}, x, {t, 0, 1}]
ers[t_] = 1 - (x[t]/. ss)[[1]]/(a Exp[t])
ers[1]
```

-0.000236

Без отключения опции `AccuracyGoal` относительная погрешность значительно больше. □

Пример. Отключим `PrecisionGoal` и решим ДУ, учитывая только абсолютную погрешность.

```
a = 10000;
si = NDSolve[{x'[t] == x[t], x[0] == a}, x, {t, 0, 1}, PrecisionGoal → ∞];
```

$\mathbf{eri[t\_] = a Exp[t] - x[t];}$
$\mathbf{eri[1]/.si}$
$\{-4.84906 \times 10^{-8}\}$
Те же вычисления без отключения опции `PrecisionGoal`.
$\mathbf{ss = NDSolve[\{x'[t] == x[t], x[0] == a\}, x, \{t, 0, 1\}];}$
$\mathbf{ers[t\_] = a Exp[t] - x[t];}$
$\mathbf{ers[1]/.ss}$
$\{-0.000217574\}$
Точное решение этого уравнения известно $a \cdot e^{-t}$**.** Абсолютная ошибка в точке `t` определяется формулой $a \cdot e^{-t} - x(t)$. Без отключения опции `PrecisionGoal` абсолютная погрешность в точке *t*=1 больше.

□

Для повышения точности решения ОДУ используется опция `WorkingPrecision`. Ее значение обычно в два раза больше значений опций `PrecisionGoal` и `AccuracyGoal`.

Пример. Рассмотрим решение ОДУ 2-го порядка, точное решение которого известно – это функция `Cos[x]`. Решим его численно с различной точностью и сравним абсолютные погрешности в точке 2π, которые отобразим в таблице.

```
TableForm[Table[
    s[i] = y /. First[NDSolve[{y''[x] + y[x] == 0, y[0] == 1, y'[0] == 0},
          y, {x, 0, 2π}, WorkingPrecision → 4(1 + i)]];
  {4(1 + i), Abs[1 - s[i][2π]], Length[s[i][[3, 1]]]},
  {i, 0, 5}],
  TableHeadings →
      {None, {"WorkingPrecision", "Error", "Number of steps"}}]
```

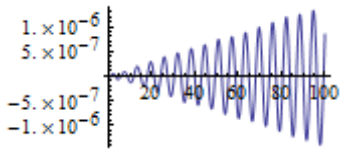
WorkingPrecision	Error	Number of steps
4	0.008	15
8	0.0008358	27
12	0.00002980281	45
16	4.41292×10^{-8}	76
20	$4.223257269 \times 10^{-10}$	89
24	$1.551461146056 \times 10^{-11}$	120

Для пояснения 3-го столбца рассмотрим следующий набор точек
$\mathbf{points = \{\{0, 0\}, \{1, 1\}, \{2, 3\}, \{3, 4\}, \{4, 3\}, \{5, 0\}\};}$
и выполним интерполяцию
$\mathbf{ifun = Interpolation[points]}$
InterpolatingFunction[{{0,5}},"<>"]
При обращении к 3-му элементу выражения `InterpolatingFunction`
$\mathbf{ifun[[3]]}$
{{0,1,2,3,4,5}}
мы получаем список первых координат точек интерполяции. Функция `Length` возвращает количество элементов этого списка. В коде примера мы печатали

это количество для каждого решения `s[i]` дифференциального уравнения, полученного при различных значениях опции `WorkingPrecision`.

Погрешность вычислений всегда растет с удалением от начальной точки

$$\mathbf{NDSolve}[\{y''[x]+y[x]==0, y[0]==1, y'[0]==0\}, y, \{x, 0, 100\}]$$
$$\mathbf{Plot}[\mathbf{Evaluate}[y[x]-\mathbf{Cos}[x]/.\%], \{x, 0, 100\}]$$

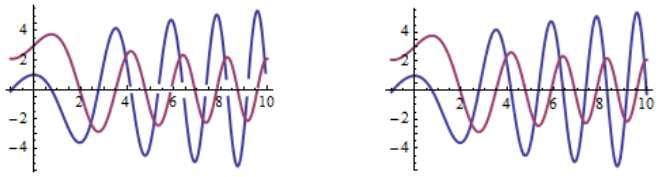


Отметим, что использование опции `WorkingPrecision` важно и в некоторых других случаях. Например, решим следующее ОДУ и построим график его решения (следующий рисунок слева)

$$\mathbf{sol}=\mathbf{DSolve}[\{x'[t]+ty[t]==0, 2y'[t]-3x[t]==0, x[0]==1, y[0]==3\}, \{x, y\}, t];$$
$$\mathbf{Plot}[\mathbf{Evaluate}[\{x[t], y[t]\}/.\mathbf{sol}], \{t, -1, 10\}]$$

Чтобы избавиться от «дыр» в графике мы увеличиваем точность внутренних вычислений (следующий рисунок справа).

$$\mathbf{Plot}[\mathbf{Evaluate}[\{x[t], y[t]\}/.\mathbf{sol}], \{t, -1, 10\}, \mathbf{WorkingPrecision}\to 20]$$



То же касается и фазовых траекторий этой задачи

$$\mathbf{ParametricPlot}[\mathbf{Evaluate}[\{\mathrm{x}[\mathrm{t}], \mathrm{y}[\mathrm{t}]\}/.\mathbf{sol}], \{\mathrm{t}, -1, 10\}]$$
$$\mathbf{ParametricPlot}[\mathbf{Evaluate}[\{x[t], y[t]\}/.\mathbf{sol}], \{t, -1, 10\}, \mathbf{WorkingPrecision}\to 20]$$

Пример. Рассмотрим уравнение.

$$\mathbf{nds}=\mathbf{NDSolve}[\{y''[x]==-y[x], y[0]==1, y'[0]==0\}, y, \{x, 0, 50\pi\}]$$

Оно нормально решается и можно построить график его решения (Это функция Cos[x]). Но если увеличить интервал, т.е. решать его в интервале $\{x, 0, 500\ \pi\}$, то мы получим сообщение

```
Maximum number of 10000 steps reached at the point x ==
1324.0481085415556`
```

Надо увеличить максимальное число шагов с помощью опции `MaxSteps`.

$$\mathbf{nds}=\mathbf{NDSolve}[\{y''[x]==-y[x], y[0]==1, y'[0]==0\}, y, \{x, 0, 500\,\pi\}, \mathbf{MaxSteps}\to 20000]$$

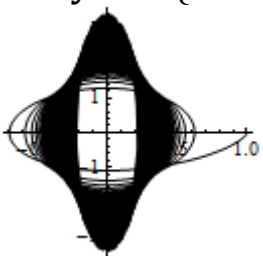
Вот еще задача, в которой следует увеличить количество шагов

$$\mathbf{nds}=\mathbf{NDSolve}[\{y''[x]+xy[x]==0, y[0]==1, y'[0]==0\}, y, \{x, 0, 200\}]$$

```
Maximum number of 10000 steps reached at the point x ==138.736390
```

Увеличим максимальное число шагов с помощью опции `MaxSteps`.

```
nds = NDSolve[{y''[x] + xy[x] == 0, y[0] == 1, y'[0] == 0},
                                        y, {x, 0, 200}, MaxSteps → ∞]
ParametricPlot[Evaluate[{y[x], y'[x]}/. nds], {x, 0, 200},
          AspectRatio → 1, PlotStyle → {Thickness[0.001], Black}]
```



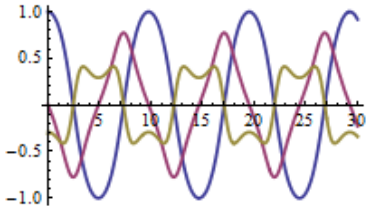
NDSolve очень сложная функция. В ней реализовано много численных алгоритмов решения ОДУ различных типов. Обычно NDSolve сама выбирает метод численного решения. Однако иногда вы будете задавать метод с помощью опции Method. Мы не ставим перед собой задачу изложения всех методов, используемых этой функцией. С ними вы можете познакомиться по справочной системе.

В версии Mathematica 9 добавлена новая функция NDSolveValue, которая в отличии от NDSolve возвращает решение не в форме правила подстановки, а в форме InterpolatingFunction функции.

```
ndsv = NDSolveValue[{y''[x] + Cos[y[x]]^2 y[x] == 0, y[0] == 1,
                                        y'[0] == 0}, y, {x, 0, 30}]
InterpolatingFunction[{{0.,30.}}," <> "]
```

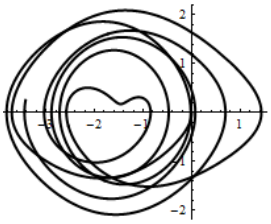
Используя функцию ndsv, можно построить график решения, а также его первой и второй производных.

```
Plot[{ndsv[x], ndsv'[x], ndsv''[x]}, {x, 0, 30}, PlotStyle → Thickness[0.01]]
```



Вот как можно построить фазовую траекторию

```
ndsv = NDSolveValue[{x''[t] + x'[t]/10 + Cos[x[t]] == Cos[t]/2,
                                        x[0] == x'[0] == 0}, x, {t, 0, 50}]
InterpolatingFunction[{{0.,50.}}," <> "]
ParametricPlot[{ndsv[t], ndsv'[t]}, {t, 0, 50}]
```



Все возможности, которые мы продемонстрировали для функции NDSolve, имеются у функции NDSolveValue.

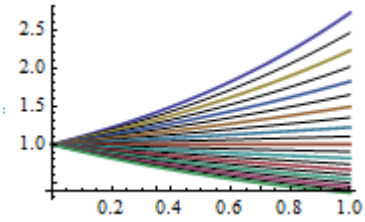
В версии Mathematica 9 появились две новые функции `ParametricNDSolve` и `ParametricNDSolveValue`. Они находят численное решение ОДУ с параметрами и отличаются одна от другой способом возврата решения.

Функция `ParametricNDSolve` возвращает решение в форме подстановки, и имеет следующий синтаксис

ParametricNDSolve[*уравнения*, y, {x,xmin,xmax}, *параметры*]

Она находит численное решение *уравнений* с *параметрами* относительно неизвестной функции *y* с независимой переменной *x*, изменяемой в диапазоне от x_{min} до x_{max}.

```
ps = ParametricNDSolve[{y'[t] + a y[t] == 0, y[0] == 1}, {y}, {t, 0, 1}, {a}]
{y → ParametricFunction[" < "" > "]}
Plot[Evaluate[Table[y[a][t] /. ps, {a, -1, 1, .1}]], {t, 0, 1}]
```



Результат возвращается в форме подстановки для искомой функции *y* `ParametricFunction` объекта. После подстановки вместо параметра конкретного значения этот объект становится `InterpolatingFunction` объектом и, фактически, функцией относительно независимой переменной

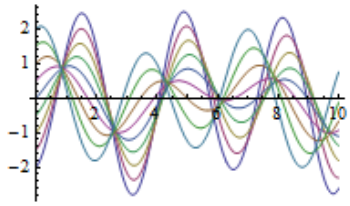
```
y1 = y[1] /. ps
InterpolatingFunction[{{0.,1.}}," <> "]
y1[0.5]
0.367879
```

При обращении к решению с параметром мы получаем «функцию двух переменных», первой из которых является параметр задачи, а второй – независимая переменная. Например, `y[1][0.5]/.ps` возвращает значение в точке *t*=0.5 решения задачи с параметром *a*=1.

Параметров в уравнении может быть несколько

```
sol = ParametricNDSolve[{y''[t] + 4 y[t] == Cos[t] + Sin[y[t]],

y[0] == a, y'[0] == b}, y, {t, 0, 10}, {a, b}]
Plot[Evaluate@Table[y[a, 1][t] /. sol, {a, -2, 2, .5}], {t, 0, 10}, PlotRange → All]
```



Функция `ParametricNDSolveValue` возвращает решение в форме функции, и имеет следующий синтаксис

ParametricNDSolveValue[*уравнения*, *выражение*, {x,xmin,xmax}, *параметры*]

Она возвращает в форме функции значение *выражения*, численно решая *уравнения* с *параметрами* и с независимой переменной *x*, изменяющейся в диапазоне от x_{min} до x_{max}. В качестве *выражения* чаще всего выступает имя искомой функции.

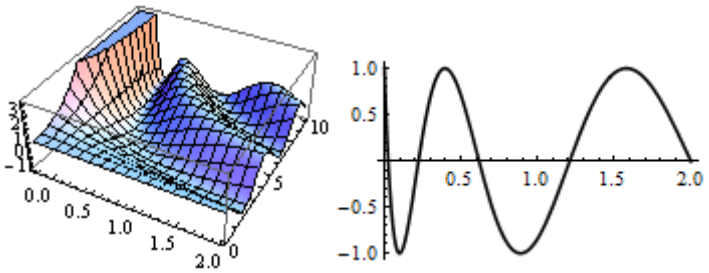
```
ps = ParametricNDSolveValue[{y''[t] + a y[t] == 0, y[0] == 1,
                             y'[0] == 0}, y, {t, 0, 10}, {a}]
ParametricFunction[" < "" > "]
```

Теперь `ps` является функцией. Мы определили параметрическую функцию (переменной *t*), зависящую от параметра *a*.

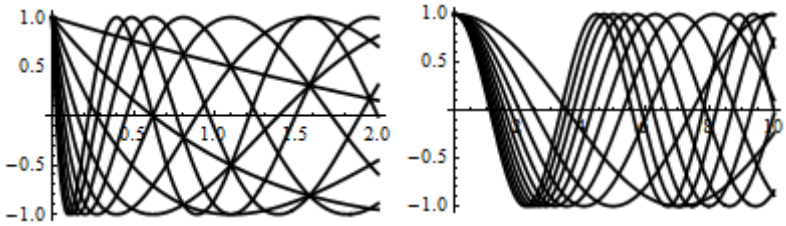
```
ps[2][0.5]
0.760245
```

Ее следует использовать так, как она была создана – каждый аргумент в своих квадратных скобках. По-другому она работать не будет. Возможные варианты ее использования приведены ниже

```
Plot3D[ps[a][t], {a, -0.25, 2}, {t, 0, 10}]
Plot[ps[a][10], {a, 0, 2}]
```

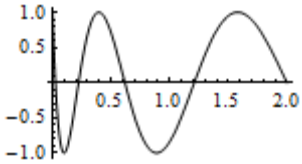


```
Plot[Table[ps[a][t], {t, 1, 10}], {a, 0, 2}]
Plot[Table[ps[a][t], {a, 0.2, 2, 0.2}], {t, 0, 10}]
```



Можно находить решение, зависящее от параметра, при фиксированном значении переменной *t*. Например

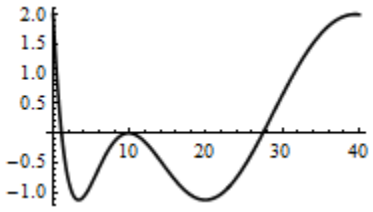
```
pfun = ParametricNDSolveValue[{y''[t] + a y[t] == 0, y[0] == 1,
                               y'[0] == 0}, y[10], {t, 0, 10}, {a}]
 Plot[pfun[a], {a, 0, 2}]
```



Фактически, это тот же график, который мы построили выше командой `Plot[ps[a][10],{a,0,2}]`.

Функция `ParametricNDSolveValue` возвращает не обязательно решение. Это может быть некоторое выражение, получаемое из искомого решения.

```
eqns = {y''[t] + a y[t] == 0, y[0] == 1, y'[0] == 0};
fun12 = ParametricNDSolveValue[eqns, y[1] + y[2], {t, 0, 15}, {a}]
Plot[fun12[a], {a, 0, 40}]
```



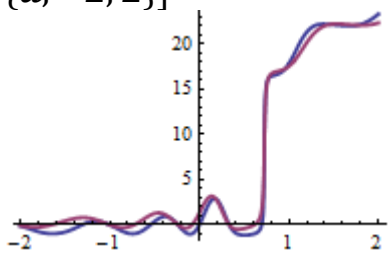
Проверим, что это то же, что y[a][1]+y[a][2]

```
fun = ParametricNDSolveValue[eqns, y, {t, 0, 15}, {a}]
Plot[fun[a][1] + fun[a][2], {a, 0, 40}]
```

Мы получим тот же график, что и выше.

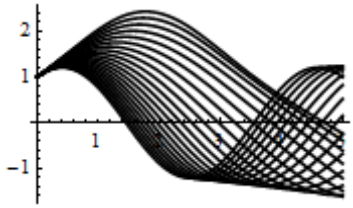
Вот еще пример.

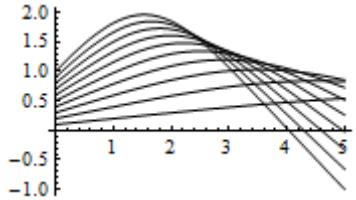
```
eqns = {y''[t] + y[t] == 3 a Sin[y[t]], y[0] == y'[0] == 1};
pfun1 = ParametricNDSolveValue[eqns, Sum[y[i], {i, 1, 10}], {t, 0, 15}, {a}];
pfun2 = ParametricNDSolveValue[eqns, Integrate[y[s], {s, 0, 10}], {t, 0, 15}, {a}];
Plot[{pfun1[a], pfun2[a]}, {a, -2, 2}]
```



Как мы говорили ранее, параметров в уравнении может быть несколько

```
ps = ParametricNDSolveValue[{y''[t] + y[t] == a Sin[y[t]],
                             y[0] == y'[0] == b}, y, {t, 0, 5}, {a, b}]
Plot[Table[ps[a, 1][t], {a, -1.5, 1.5, .15}], {t, 0, 5}, PlotRange → All]
Plot[Table[ps[1, b][t], {b, 0, 1, .1}], {t, 0, 5}]
```

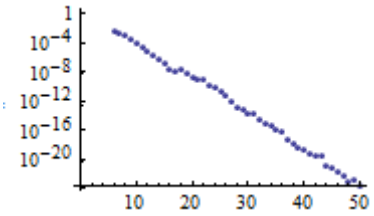




Функция `ParametricNDSolveValue` имеет такие же опции как и функция `NDSolve`.

Вот еще один пример ее использования. Численно решим задачу $x'(t)=x(t)$, $x(0)=1$ для различных значений `WorkingPrecision` и графически изобразим абсолютную погрешность вычислений в точке t=1, равную $|x(1)-e|$

```
err = ParametricNDSolveValue[{x'[t] == x[t], x[0] == 1}, Abs[x[1] - E],
                             {t, 0, 1}, {wp}, WorkingPrecision → wp];
ListLogPlot[Table[{wp, err[wp]}, {wp, 6, 50}]]
```

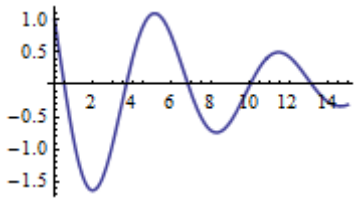


Пример. Найдем начальное условие $x'(0)$ при котором решение $x(t)$ ДУ обратиться в 0 при t=10, т.е. $x(10)=0$

```
pf = ParametricNDSolveValue[{x''[t] + 0.25x'[t] + x[t] == 0,
                             x[0] == 1, x'[0] == s}, x, {t, 0, 15}, {s}];
root = FindRoot[pf[s][10] == 0, {s, 11}]
{s→-1.95507}
```

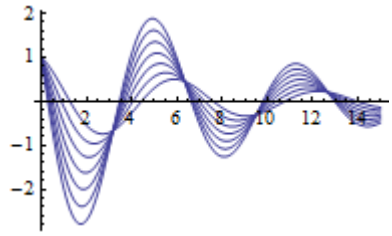
Для проверки построим график решения при этом значении параметра

```
Plot[pf[s /. root][t], {t, 0, 15}, PlotRange → All]
```



Сравним поведение кривых при близких значениях параметра s.

```
s0 = s /. root;
sL = Table[s0 + i * 0.5, {i, -3, 3}]
{-3.455, -2.955, -2.455, -1.95507, -1.455, -0.955, -0.455}
Plot[Table[pf[s][t], {s, sL}], {t, 0, 15}, PlotRange → All]
```



4.2.2 Численное решение систем ОДУ

Системы ОДУ численно решаются с помощью тех же функций, которые используются для решения одного ОДУ. Начиная с «древних версий» системы *Mathematica* для этого используется функция `NDSolve`.

```
sol = NDSolve[{x'[t] == y[t], y'[t] == -0.01y[t] - Sin[x[t]],
               x[0] == 0, y[0] == 2.1}, {x, y}, {t, 0, 1}]
{{x → InterpolatingFunction[{{0.,1.}},"<>"],
  y → InterpolatingFunction[{{0.,1.}},"<>"]}}
```

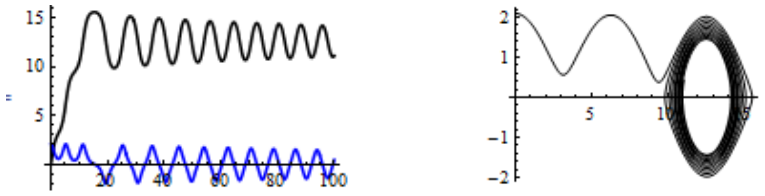
Обратите внимание что, мы имеем единственное решение в виде двух правил подстановки для `x` и `y` соответственно. Чтобы получить из них две функции достаточно выполнить следующие команды

```
x = x /. sol[[1]]
y = y /. sol[[1]]
```

InterpolatingFunction[{{0.,100.}},"<>"]
InterpolatingFunction[{{0.,100.}},"<>"]

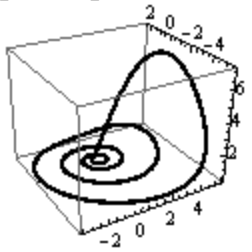
Как видно из последнего решения, x и y являются уже функциями и для них можно применять обычные математические действия и строить график.

```
Plot[{x[t], y[t]}, {t, 0, 100}, PlotStyle →
  {{Thickness[0.01], RGBColor[0, 0, 0]}, {Thickness[0.01], RGBColor[0, 0, 1]}}]
ParametricPlot[{x[t], y[t]}, {t, 0, 100}, PlotStyle →
                    {RGBColor[0, 0, 0]}, AspectRatio → 0.7]
```



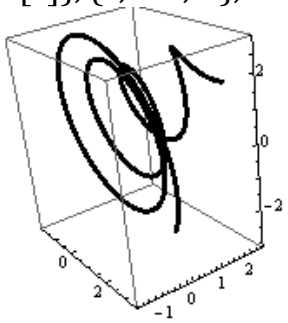
Вот пример построения фазовой траектории системы 3 – х ОДУ без создания функций решения

```
Remove[x, y, z, t]
sol = NDSolve[{x'[t] == −y[t] − z[t], x[0] == −0.04,
                y'[t] == x[t] + 0.425y[t], y[0] == −0.3,
                z'[t] == 2 − z[t](4 − x[t]), z[0] == 0.52},
      {x, y, z}, {t, 0, 25}]
ParametricPlot3D[Evaluate[{x[t], y[t], z[t]}/. sol], {t, 0, 25},
      PlotStyle → {Thickness[0.01], RGBColor[0, 0, 0]}, PlotRange → All]
```



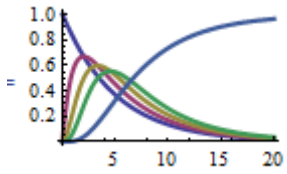
Вот пример построения фазовой траектории системы 3 – х ОДУ с созданием функций решения

```
Remove[x, y, z, t]
nds = NDSolve[{x'[t] == y[t] − z[t], y'[t] == z[t] − x[t], z'[t] =
          = x[t] − 2y[t], x[0] == 1, y[0] == 0, z[0] =
          = 2}, {x, y, z}, {t, −4, 8}]
x = nds[[1, 1, 2]];  y = nds[[1, 2, 2]];  z = nds[[1, 3, 2]]
ParametricPlot3D[{x[t], y[t], z[t]}, {t, −4, 8}, PlotStyle → Thickness[0.01]]
```



Неизвестные функции задачи не обязаны быть представлены уникальными идентификаторами (именами). Когда неизвестных функций много вам, вероятно, будет удобно обозначать их, например так, `y[i]`.

$$\mathbf{eqns} = \mathbf{Join}[\mathbf{Table}[\boldsymbol{y}[\boldsymbol{i}]'[\boldsymbol{x}] == \boldsymbol{y}[\boldsymbol{i}-\mathbf{1}][\boldsymbol{x}] - \boldsymbol{y}[\boldsymbol{i}][\boldsymbol{x}], \{\boldsymbol{i}, \mathbf{2}, \mathbf{4}\}],$$
$$\{\boldsymbol{y}[\mathbf{1}]'[\boldsymbol{x}] == -\mathbf{0.2}\boldsymbol{y}[\mathbf{1}][\boldsymbol{x}], \boldsymbol{y}[\mathbf{5}]'[\boldsymbol{x}] == \mathbf{0.2}\boldsymbol{y}[\mathbf{4}][\boldsymbol{x}],$$
$$\boldsymbol{y}[\mathbf{1}][\mathbf{0}] == \mathbf{1}\}, \mathbf{Table}[\boldsymbol{y}[\boldsymbol{i}][\mathbf{0}] == \mathbf{0}, \{\boldsymbol{i}, \mathbf{2}, \mathbf{5}\}]];$$
$$\mathbf{nds} = \mathbf{NDSolve}[\mathbf{eqns}, \mathbf{Table}[\boldsymbol{y}[\boldsymbol{i}], \{\boldsymbol{i}, \mathbf{5}\}], \{\boldsymbol{x}, \mathbf{0}, \mathbf{20}\}];$$
$$\mathbf{Plot}[\mathbf{Evaluate}[\mathbf{Table}[\boldsymbol{y}[\boldsymbol{i}][\boldsymbol{x}], \{\boldsymbol{i}, \mathbf{1}, \mathbf{5}\}]/.\mathbf{nds}], \{\boldsymbol{x}, \mathbf{0}, \mathbf{20}\}]$$



Функция `NDSolve` может рассматривать обозначение `y[x]` как обозначение **вектор – функции**, если для `y[x]` представить вектор инициализаций, например, `y[0]=={v1,v2,...,vn}`. Для обращения к компонентам такой вектор функции следует выбирать выражения `y[x][[1]], y[x][[2]]`,...

Пример. Одну и ту же задачу о колебании линейного осциллятора можно записать и решить в *Mathematica* тремя способами.

1. Как систему ОДУ 1 – го порядка

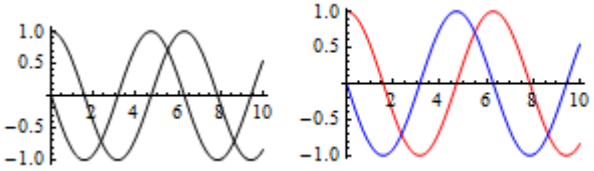
$$\mathbf{NDSolve}[\{\boldsymbol{x}'[\boldsymbol{t}] == \boldsymbol{y}[\boldsymbol{t}], \boldsymbol{y}'[\boldsymbol{t}] == -\boldsymbol{x}[\boldsymbol{t}], \boldsymbol{x}[\mathbf{0}] == \mathbf{1},$$
$$\boldsymbol{y}[\mathbf{0}] == \mathbf{0}\}, \{\boldsymbol{x}, \boldsymbol{y}\}, \{\boldsymbol{t}, \mathbf{0}, \mathbf{10}\}]$$

2. Как ОДУ 2 – го порядка

$$\mathbf{NDSolve}[\{\boldsymbol{y}''[\boldsymbol{x}] + \boldsymbol{y}[\boldsymbol{x}] == \mathbf{0}, \boldsymbol{y}[\mathbf{0}] == \mathbf{1}, \boldsymbol{y}'[\mathbf{0}] == \mathbf{0}\}, \boldsymbol{y}, \{\boldsymbol{x}, \mathbf{0}, \mathbf{10}\}]$$

3. Как задачу для вектор функции

$$\mathbf{nds} = \mathbf{NDSolve}[\{\boldsymbol{z}'[\boldsymbol{t}] == \{\{\mathbf{0}, \mathbf{1}\}, \{-\mathbf{1}, \mathbf{0}\}\}.\boldsymbol{z}[\boldsymbol{t}], \boldsymbol{z}[\mathbf{0}] == \{\mathbf{1}, \mathbf{0}\}\}, \boldsymbol{z}, \{\boldsymbol{t}, \mathbf{0}, \mathbf{10}\}]$$
$$\mathbf{Plot}[\boldsymbol{z}[\boldsymbol{t}]/.\mathbf{First}[\mathbf{nds}], \{\boldsymbol{t}, \mathbf{0}, \mathbf{10}\}]$$



Чтобы обратиться к каждой компоненте вектор функции как к отдельной функции можно выполнить команды

$$\boldsymbol{g} = \mathbf{nds}[[\mathbf{1}, \mathbf{1}, \mathbf{2}]]$$
$$\mathbf{Plot}[\{\boldsymbol{g}[\boldsymbol{x}][[\mathbf{1}]], \boldsymbol{g}[\boldsymbol{x}][[\mathbf{2}]]\}, \{\boldsymbol{x}, \mathbf{0}, \mathbf{10}\}, \mathbf{PlotStyle} \to \{\{\mathbf{Red}\}, \{\mathbf{Blue}\}\}]$$

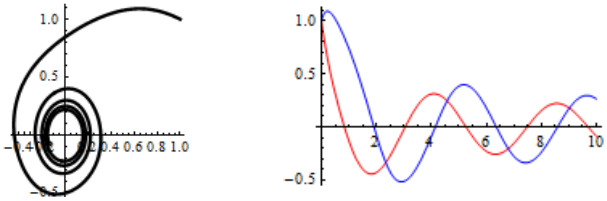
График, построенный последней командой, показан на предыдущем рисунке справа. Обратите внимание, что при таком построении вы можете каждой кривой задать свой стиль/цвет.

□

Пример. Иногда удается записать нелинейную систему ОДУ как задачу относительно вектор – функции. При этом, например, запись $\boldsymbol{x}[\boldsymbol{t}]^{\mathbf{3}}$ будет означать покомпонентное возведение в степень, а запись $\{\{\mathbf{0}, -\mathbf{1}\}, \{\mathbf{2}, \mathbf{0}\}\}.\boldsymbol{x}[\boldsymbol{t}]$ – «матричное» умножение (функция `Dot` – точка).

$$\boldsymbol{s} = \mathbf{NDSolve}[\{\boldsymbol{x}'[\boldsymbol{t}] == \{\{\mathbf{0}, -\mathbf{1}\}, \{\mathbf{2}, \mathbf{0}\}\}.\boldsymbol{x}[\boldsymbol{t}] - \boldsymbol{x}[\boldsymbol{t}]^{\mathbf{3}}, \boldsymbol{x}[\mathbf{0}] == \{\mathbf{1}, \mathbf{1}\}\}, \boldsymbol{x}, \{\boldsymbol{t}, \mathbf{20}\}]$$

$$\mathbf{ParametricPlot[Evaluate[\{x[t]\}/.s],\{t,0,20\}]}$$
$$\mathbf{g=s[[1,1,2]]}$$
$$\mathbf{Plot[\{g[x][[1]],g[x][[2]]\},\{x,0,10\},PlotStyle\rightarrow\{\{Red\},\{Blue\}\}]}$$



Та же задача может быть записана как обычная система ОДУ 1 – го порядка

$$\mathbf{s=NDSolve[\{x'[t]==-y[t]-x[t]^3,y'[t]==2x[t]-y[t]^3,}$$
$$\mathbf{x[0]==y[0]==1\},\{x,y\},\{t,20\}];}$$

□

Вот система ОДУ 2 – го порядка, записанная относительно вектор – функции

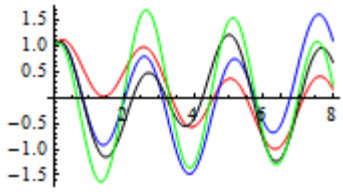
$$\mathbf{s=NDSolve[\{y''[x]+\begin{pmatrix}1&1&1&1\\1&2&1&2\\1&1&3&1\\1&2&1&4\end{pmatrix}.y[x]==0,}$$
$$\mathbf{y[0]==y'[0]==\{1,1,1,1\}\},y,\{x,0,8\}];}$$

$\{\{y\rightarrow$ InterpolatingFunction[{{0.,8.}},"<>"]$\}\}$

$$\mathbf{y[5]/.s}$$

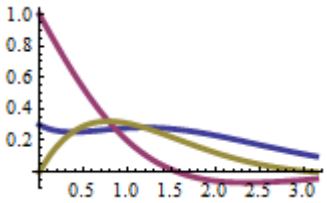
{{0.36310, 0.658461, 1.215740, 1.4692304}}

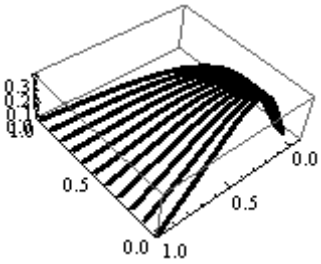
$$\mathbf{g=s[[1,1,2]]}$$
$$\mathbf{Plot[\{g[x][[1]],g[x][[2]],g[x][[3]],g[x][[4]]\},\{x,0,8\},}$$
$$\mathbf{PlotStyle\rightarrow\{\{Red\},\{Blue\},\{Black\},\{Green\}\}]}$$



В следующем примере мы строим несколько фазовых траекторий системы ДУ, соответствующих разным начальным условиям, используя функцию `ParametricNDSolve`.

$$\mathbf{Remove[x,y,z,t,X,Y,Z]}$$
$$\mathbf{nds=ParametricNDSolve[\{x'[t]==-x[t]+z[t],x[0]==0.1a,}$$
$$\mathbf{y'[t]==-y[t]-z[t],y[0]==1,}$$
$$\mathbf{z'[t]==y[t]-z[t],z[0]==0\},}$$
$$\mathbf{\{x,y,z\},\{t,0,\pi\},\{a\}]}$$
$$\mathbf{X[a\_]=x[a]/.nds}$$
$$\mathbf{Y[a\_]=y[a]/.nds}$$
$$\mathbf{Z[a\_]=z[a]/.nds}$$
$$\mathbf{Plot[\{X[3][t],Y[3][t],Z[3][t]\},\{t,0,\pi\}]}$$
$$\mathbf{ParametricPlot3D[Table[\{X[a][t],Y[a][t],Z[a][t]\},\{a,0,10\}],\{t,0,\pi\}]}$$





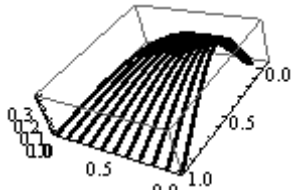
Пример. Функция `ParametricNDSolve` также может работать с вектор – функциями.

$$\mathbf{s = ParametricNDSolve[\{x'[t] == \{\{-1,0,1\},\{0,-1,-1\},\{0,1,-1\}\}.x[t],\ x[0] == \{0.1a,1,0\}\},x,\{t,0,\pi\},\{a\}]}$$

$\{x \to \text{ParametricFunction[" < "" > "]}\}$

$$\boldsymbol{g = s[[1,2]]}$$

$$\mathbf{ParametricPlot3D[Table[}g[a][t],\{a,0,10\}],\{t,0,\pi\}]$$



Соответствующую систему ОДУ 1 – го порядка относительно 3 – х неизвестных функций мы решали ранее в этом параграфе.

Также как и для одного ОДУ для решения систем ОДУ в Mathematica 9 можно использовать функцию `NDSolveValue`.

Пример. Исследуем решение задачи Коши для системы уравнений

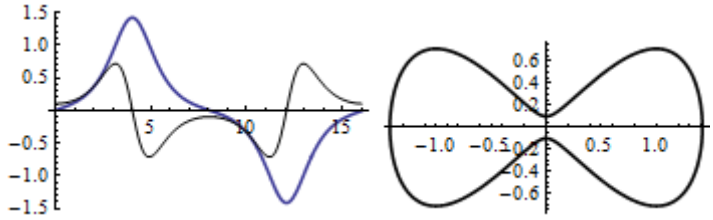
$$x' = y(t),\ y' = -x^3(t) + x(t),\ x(0) = 0,\ y(0) = 0.1$$

Имеем

$$\boldsymbol{s = NDSolveValue[\{x'[t] == y[t], y'[t] == x[t] - x[t]^3,\ x[0] == 0, y[0] == 0.1\},\{x,y\},\{t,0,16\}]}$$

$$\mathbf{Plot[\{}s[[1]][t], s[[2]][t]\},\{t,0,16\}]$$

$$\mathbf{ParametricPlot[\{}s[[1]][t], s[[2]][t]\},\{t,0,16\}]$$



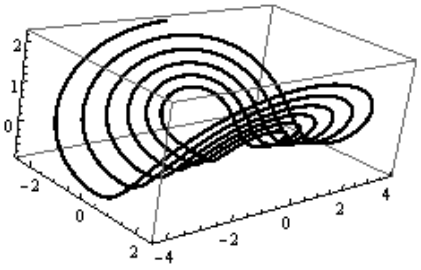
На левом рисунке показаны графики функций `x(t)`, `y(t)`, а на правом – фазовая траектория.

Пример. Решим систему дифференциальных уравнений

$$\frac{dx}{dt} = y,\ \frac{dy}{dt} = 0.1y - x\cdot(z-1) - x^3,\ \frac{dz}{dt} = x\cdot y - 0.1z$$

с начальными условиями $x(0) = 1,\ y(0) = 1,\ z(0) = 0$ и построим ее фазовый портрет.

```
s = NDSolveValue[{x'[t] == y[t], y'[t] == 0.1y[t] - x[t](z[t] - 1) - x[t]^3,
    z'[t] == x[t]y[t] - 0.1z[t], x[0] == 1, y[0] == 1, z[0] == 0},
                                                {x, y, z}, {t, 0, 25}]
ParametricPlot3D[{s[[1]][t], s[[2]][t], s[[3]][t]}, {t, 0, 25}]
```

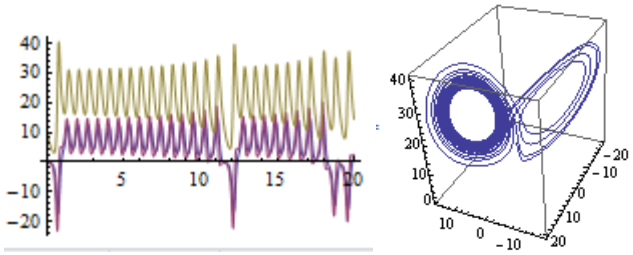


Пример. Решим систему уравнений

$$\begin{cases} x' = s \cdot (y - x) \\ y' = x \cdot (r - z) - y, \\ z' = x \cdot y - b \cdot z \end{cases}$$

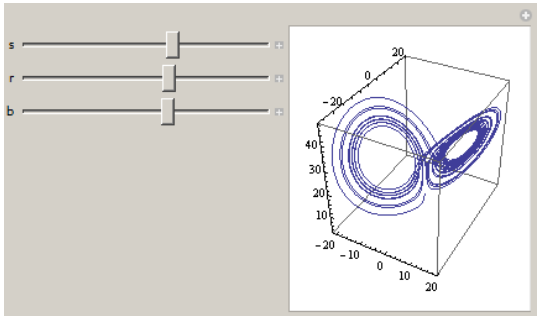
где s, r, b некоторые параметры.

```
s = 10; r = 25; b = 3;
sys = {x'[t] == s(y[t] - x[t]), y'[t] == x[t](r - z[t]) - y[t],
    z'[t] == x[t]y[t] - bz[t], x[0] == 1, y[0] == -1, z[0] == 10};
nds = NDSolveValue[sys, {x, y, z}, {t, 0, 20}]
Plot[{nds[[1]][t], nds[[2]][t], nds[[3]][t]}, {t, 0, 20}]
ParametricPlot3D[{nds[[1]][t], nds[[2]][t], nds[[3]][t]}, {t, 0, 20}]
```



Влияние параметров ***s, r, b*** можно исследовать по поведению фазовых траекторий в блоке `Manipulate`.

```
Manipulate[
  sys = {x'[t] == s(y[t] - x[t]), y'[t] == x[t](r - z[t]) - y[t],
  z'[t] == x[t]y[t] - bz[t], x[0] == 1, y[0] == -1, z[0] == 10};
  nds = NDSolveValue[sys, {x, y, z}, {t, 0, 20}];
  ParametricPlot3D[{nds[[1]][t], nds[[2]][t], nds[[3]][t]}, {t, 0, 20},
                                  PlotRange → All],
  {{s, 10}, 1, 20}, {{r, 25}, 5, 45}, {{b, 3}, 1, 5},
  ControlPlacement → Left, AutoAction → False]
```

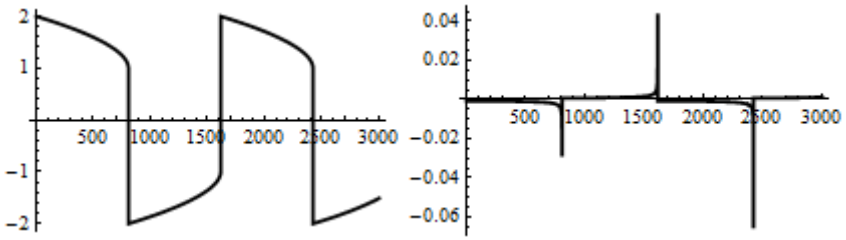


□

Обычно функция NDSolve сама выбирает метод численного решения. В справочной системе *Mathematica* приводится уравнение Ван-дер-Поля $z''+z-K\cdot(1-z^2)\cdot z'=0$, решение которого обычными методами типа Рунге – Кутта дает неудовлетворительный результат. Запишем уравнение в виде системы (полагаем K=1000)

$$\begin{cases} x'=y \\ y'=-x+K\cdot(1-x^2)\cdot y \end{cases}, \quad \begin{pmatrix} x(0) \\ y(0) \end{pmatrix}=\begin{pmatrix} 2 \\ 0 \end{pmatrix}$$

```
vdp = {x'[t] == y[t], y'[t] == -x[t] + 1000(1 - x[t]^2) * y[t],
                                    x[0] == 2, y[0] == 0};
sol = NDSolve[vdp, {x, y}, {t, 3000}]
{{x → InterpolatingFunction[{{0.,3000.}}," <> "],
  y → InterpolatingFunction[{{0.,3000.}}," <> "]}}
Plot[Evaluate[x[t]/.sol], {t, 0, 3000}, PlotRange → All]
Plot[Evaluate[y[t]/.sol], {t, 0, 3000}, PlotRange → All]
```



Система *Mathematica* сама правильно выбрала метод решения. Дело в том, что эта задача является примером так называемых жестких систем, для решения которых в *Mathematica* используются специальные методы решения. Методы не предназначенные для таких задач, вероятно, не построят решение

```
NDSolve[{x'[t] == y[t], y'[t] == -x[t] + 1000(1 - x[t]^2)y[t],
   x[0] == 2, y[0] == 0}, {x, y}, {t, 2000}, Method → "ExplicitRungeKutta"]
NDSolve::ndstf: At t==0.008522, system appears to be
stiff. Methods Automatic, BDF, or StiffnessSwitching may
be more appropriate.
```

4.3 Представление решений ОДУ в системе *Mathematica*

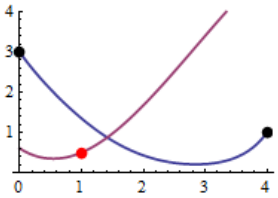
Наглядное графическое представление решений ОДУ и их систем является существенным элементом исследования. В предыдущих параграфах этого пособия мы показали как строить графики решений, фазовые траектории и

анимацию. Здесь мы приведем примеры использования других средств системы *Mathematica*, которые также помогут быть полезны при изучении поведения решений ОДУ.

1^0. Напомним об опциях `Epilog` и `Prolog`, которые имеют функции построения графиков. Их значениями могут быть графические примитивы, которые отображаются в области графика после или до его создания.

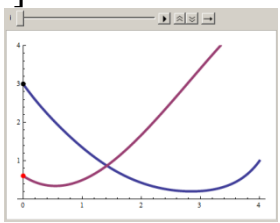
В следующем примере вместе с графиком решений мы отображаем точки дополнительных условий.

```
sol = NDSolve[{x''[t] == y[t]x[t], y'[t] == 2 - x[t],
                        x[0] == 3, x[4] == 1, y[1] == 0.5}, {x, y}, t]
Plot[Evaluate[{x[t], y[t]}/. sol[[1]]], {t, 0, 4},
Epilog → {PointSize[Large], Point[{0, 3}], Point[{4, 1}], Red, Point[{1, 0.5}]},
PlotRange → {{-0.1, 4.1}, {0, 4}}, PlotStyle → Thickness[0.01]]
```



Примитив, используемый в `Epilog`, можно менять/перемещать вместе с параметром анимации.

```
Remove[x, y]
sol = NDSolve[{x''[t] == y[t]x[t], y'[t] == 2 - x[t],
                        x[0] == 3, x[4] == 1, y[1] == 0.5}, {x, y}, t]
x = sol[[1, 1, 2]]; y = sol[[1, 2, 2]];
Animate[
  Plot[Evaluate[{x[t], y[t]}], {t, 0, 4},
    PlotStyle → Thickness[0.01], PlotRange → {{-0.1, 4.1}, {0, 4}},
    Epilog → {PointSize[Large], Point[{i, x[i]}],
                Red, Point[{i, y[i]}]}], {i, 0, 4},
  AnimationRunning → False]
```



2^0. Практически любой управляющий объект (`Control`) можно использовать для улучшения наглядности представления решений ОДУ и их систем. Здесь мы продемонстрируем использование некоторых из них.

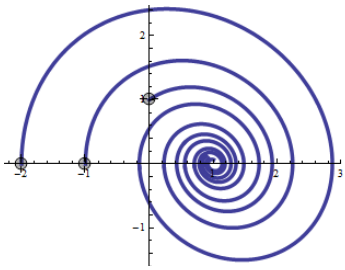
Функция `LocatorPane` предоставляет область с «локатором», который можно перемещать с помощью мыши. В формате

```
LocatorPane[Dynamic[pt], тело]
```

она позволяет динамически менять положение указателя и использовать его координаты в теле (последовательности команд).

В следующем примере мы создаем 3 локатора, координаты которых используются для построения трех фазовых траекторий, получаемых при трех различных начальных условиях.

```
xmax = 20;
DynamicModule[{pt = {{-1, 0}, {0, 1}, {-2, 0}}, dd, p, pp},
  LocatorPane[Dynamic[pt],
    Dynamic[
      pp = Table[
            dd = DSolve[{y''[x] + 0.3y'[x] + y[x] == 1,
                  y[0] == pt[[i, 1]], y'[0] == pt[[i, 2]]}, y, x];
             p = ParametricPlot[Evaluate[{y[x]/. dd[[1]],
                  y'[x]/. dd[[1]]}], {x, 0, xmax},
                  PlotRange → {{-3, 3}, {-3, 3}}];
            p, {i, 1, 3}];
      Show[pp]]]]
```

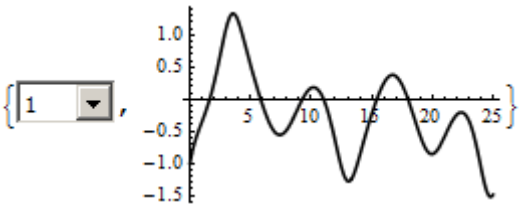


«Захватите» мышью «локатор» и перемещайте его. Каждое новое положение «локатора»определяет новые начальные значения и, следовательно, новую фазовую траекторию системы. Кривые будут динамически перерисовываться.

□

С помощью элемента «выпадающий список» (создается функцией `PopupMenu`) можно менять значение какого-либо параметра в ДУ и автоматически получать новое решение.

```
DynamicModule[{a},
  {PopupMenu[Dynamic[a], {1 → 1, 2 → 1.25, 3 → 1.5, 4 → 1.75, 5 → 2}],
   Dynamic[
      s = NDSolve[{x''[t] + x'[t] - x[t] + x[t]^3 == Sin[t],
                   x[0] == -1, x'[0] == a}, x, {t, 0, 25}];
      Plot[Evaluate[x[t]/. s], {t, 0, 25}]]}]
```



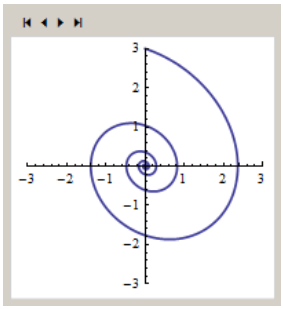
Выбор в списке значения автоматически перестраивает график решения.

□

Элемент `SlideView` (функция, которая его создает) принимает список объектов и представляет один из них в документе. Щелчки по кнопкам  элемента позволяют переходить последовательно от представления одного

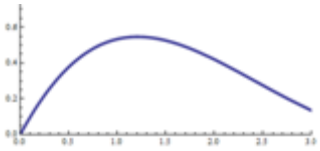
объекта к другому. Отображаемым объектом может быть, например, график решения или фазовая траектория.

```
SlideView[
    Table[
        ds = DSolve[{x'[t] == y[t], y'[t] == -x[t] - y[t]/a,
                                    x[0] == 0, y[0] == a}, {x, y}, t];
        p = ParametricPlot[
            Evaluate[{x[t]/. ds[[1, 1]], y[t]/. ds[[1, 2]]}], {t, 0, 100},
            PlotRange → {{-3, 3}, {-3, 3}}, PlotStyle → Thickness[0.01]];
    p, {a, 1, 4}]]
```



Элемент `FlipView` (одноименная функция, которая его создает) тоже может представлять один объект из нескольких, заданных в списке. Только переход от одного объекта к другому выполняется при щелчке мышью по объекту, например, по графику решения.

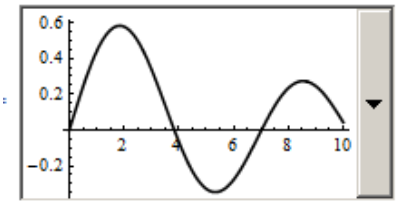
```
ds = DSolve[{x'[t] == y[t], y'[t] == -x[t] - y[t],
                                x[0] == 0, y[0] == 1}, {x, y}, t];
FlipView[
  {ParametricPlot[Evaluate[{x[t]/. ds[[1, 1]], y[t]/. ds[[1, 2]]}], {t, 0, 100},
                                        PlotRange → {{-1, 1}, {-1, 1}}],
   Plot[Evaluate[x[t]/. ds[[1, 1]]], {t, 0, 100}, PlotRange → {{0, 3}, {0, 1}}],
    Plot[Evaluate[y[t] /. ds[[1, 2]]], {t, 0, 100}, PlotRange → {{0, 3}, {-1, 1}}],
  }]
```

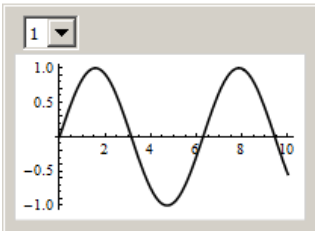


Щелкая мышью по области рисунка вы можете переходить последовательно от одного графика к другому.

Аналогичным свойством обладает элемент `PopupView`, `MenuView`. Они принимают список объектов, например, графических и по очереди отображают их в своей рабочей области.

```
PopupView[Table[Plot[BesselJ[n, x], {x, 0, 10}], {n, 3}]]
MenuView[Table[Plot[Sin[nx], {x, 0, 10}], {n, 5}]]
```

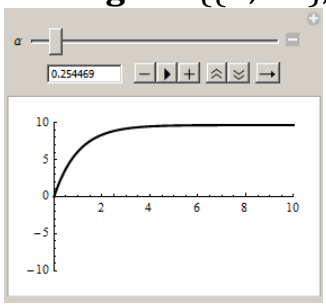




Первый из этих элементов показан на предыдущем рисунке слева, второй – справа. Внутри списков, генерируемых функцией `Table`, вы можете поставить любые, имеющие экранное представление функции, например, графики решений ОДУ или таблицы значений решений.

Однако, наиболее часто используемым управляющим объектом, вероятно, является `Manipulate`.

```
Remove[z, x, y]
sol = DSolve[{D[x[t, α], {t, 2}] == −D[x[t, α], t],
              x[0, α] == 0, Derivative[1, 0][x][0, α] == 10Cos[α]}, x[t, α], t]
z[t_, α_] = sol[[1, 1, 2]]
{{x[t, α] → 10e^−t (−1 + e^t) Cos[α]}}
Manipulate[
    Plot[z[t, α], {t, 0, 10}, PlotRange → {{0, 10}, {−10, 10}}], {α, 0, π}]
```



Перемещая бегунок манипулятора, мы меняем значение параметра α и, тем самым, строим график решения $z(t,\alpha)$ при другом α. Раскрыв значок «+» справа от бегунка, можно наблюдать за значением параметра α.

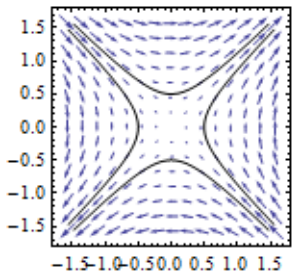
3^0. Кроме графиков решений и фазовых траекторий существуют другие способы улучшенного наглядного представления решений ОДУ. Так фазовые траектории иногда удобно рисовать поверх векторных полей, которые в системе *Mathematica* можно строить различными способами. Напомним некоторые понятия, связанные с векторными полями на плоскости.

Если каждой точке (x,y) двумерного пространства поставить в соответствие вектор $\mathbf{v}(x,y)=(P(x,y),Q(x,y))$, в результате получим векторное поле. Один из способов его визуализации состоит в рисовании в некотором множестве точек плоскости стрелок, представляющих в некотором масштабе значение вектора $\boldsymbol{v}(x,y)$ в этих точках. Такой рисунок в *Mathematica* создается функцией `VectorPlot`, которой в качестве аргументов передаются скалярные функции $P(x,y),Q(x,y)$. Кроме того, для наглядного представления векторных полей используются линии тока (векторные линии, силовые линии). Через каждую точку плоскости проходит одна линия. За исключением точек, где поле не определено или $\boldsymbol{v}(x,y)=(0,0)$, линии тока никогда не пересекаются.

В декартовых координатах дифференциальные уравнения линий тока имеют вид $\frac{dx}{P(x,y)}=\frac{dy}{Q(x,y)}$. Поле линий тока в *Mathematica* строится функцией `StreamPlot`.

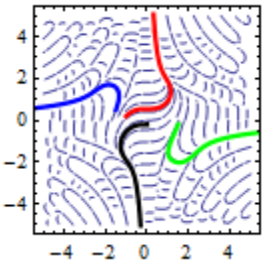
Пример. Построим несколько фазовых траекторий системы двух ДУ и векторное поле , соответствующее этой системе.

```
sys = {x'[t] == y[t], y'[t] == x[t]};
bc = {{x[0] == 0, y[0] == 0.5}, {x[0] == 0.5, y[0] == 0},
      {x[0] == 0, y[0] == -0.5}, {x[0] == -0.5, y[0] == 0}};
pp = Table[
        se = Join[sys, bc[[i]]];
        u = DSolve[se, {x, y}, t];
        p = ParametricPlot[{x[t]/.u[[1, 1]], y[t]/.u[[1, 2]]}, {t, -1.8, 1.8}];
        p, {i, 1, 4}];
vp = VectorPlot[{y, x}, {x, -π/2, π/2}, {y, -π/2, π/2}];
Show[vp, pp]
```



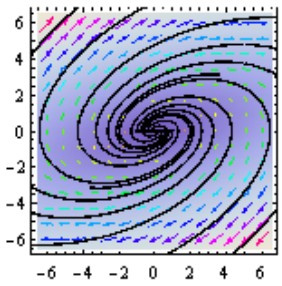
Пример. В следующем примере мы строим поле линий тока векторного поля и некоторые из таких линий, которые получаются из решения соответствующей системы ОДУ при различных начальных условиях.

```
sys = {x'[t] == y[t]Cos[x[t]y[t]], y'[t] == x[t]Sin[x[t]y[t]]};
col = {Red, Blue, Black, Green};
bc = {{x[0] == -1, y[0] == 0.2}, {x[0] == -1.5, y[0] == 0.5},
      {x[0] == 0, y[0] == -0.2}, {x[0] == 1.5, y[0] == -0.2}};
pp = Table[
        se = Join[sys, bc[[i]]];
        u = NDSolve[se, {x, y}, {t, 0, 12}];
        p = ParametricPlot[{x[t]/.u[[1, 1]], y[t]/.u[[1, 2]]}, {t, 0, 12},
                           PlotStyle → {Thickness[0.02], col[[i]]}];
        p, {i, 1, 4}];
vp = StreamPlot[{yCos[xy], xSin[xy]}, {x, -5, 5}, {y, -5, 5}];
Show[vp, pp]
```



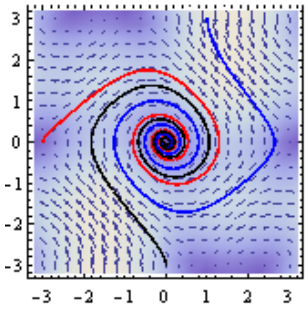
Пример. Большую наглядность дает функция `VectorDensityPlot`, которая кроме векторного поля строит плотность некоторого скалярного поля, например, поля длины/нормы векторного поля. Она раскрашивает область в цвета, яркость которых соответствует значению этого скалярного поля.

```
sys = {x'[t] == 2y[t], y'[t] == -x[t] + y[t]};
pp = Table[
        se = Join[sys, {x[0] == -1 + a, y[0] == 0}];
        u = DSolve[se, {x, y}, t];
        p = ParametricPlot[{x[t]/.u[[1, 1]], y[t]/.u[[1, 2]]}, {t, 0, 2π}];
        p, {a, 0, 2, 0.2}];
vp = VectorDensityPlot[{2y, -x + y}, {x, -6, 6}, {y, -6, 6},
                                    VectorColorFunction → Hue];
Show[vp, pp, AspectRatio → Automatic]
```



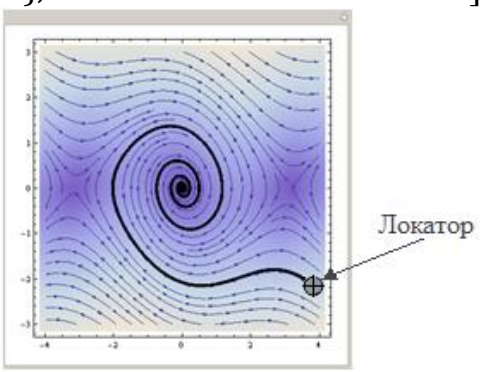
Пример. На фоне векторного поля и его нормы построим несколько линий тока этого векторного поля.

```
sys = {x'[t] == Sin[y[t]], y'[t] == -Sin[x[t]] - y[t]/4};
bc = {{x[0] == -3, y[0] == 0}, {x[0] == 1, y[0] == 3},
                                {x[0] == 0, y[0] == -3}};
col = {Red, Blue, Black};
pp = Table[
        se = Join[sys, bc[[i]]];
        u = NDSolve[se, {x, y}, {t, 0, 30}];
        p = ParametricPlot[{x[t]/.u[[1, 1]], y[t]/.u[[1, 2]]}, {t, 0, 30},
                                PlotStyle-> {Thickness[0.01], col[[i]]}];
        p, {i, 1, 3}];
vp = VectorDensityPlot[{Sin[y], -Sin[x] - y/4}, {x, -3, 3}, {y, -3, 3},
                                                VectorPoints → 21];
Show[vp, pp]
```



Пример. В манипуляторе можно использовать объект, который представляет «локатор» - точку, которую можно перемещать с помощью мыши. В нашем примере координаты «локатора» будут использоваться для задания начальных значений системы ОДУ, определяющей фазовую траекторию

```
Remove[x, y]; T = 100;
vp = StreamDensityPlot[{y, -Sin[x] - .25y}, {x, -4, 4}, {y, -3, 3}];
Manipulate[
    u = NDSolve[{x'[t] == y[t], y'[t] == -Sin[x[t]] - .25y[t],
                 x[0] == pnt[[1]], y[0] == pnt[[2]]}, {x, y}, {t, 0, T}];
    p1 = ParametricPlot[Evaluate[{u[[1, 1, 2]][t], u[[1, 2, 2]][t]}], {t, 0, T},
                        PlotRange → {{-2, 2}, {-2, 2}}];
    Show[vp, p1],
    {{pnt, {1, 0}}, Locator}, SaveDefinitions → True]
```



«Захватите» мышью «локатор» и перемещайте его. Каждое новое положение «локатора» задает новые начальные значения и, следовательно, новую линию тока (фазовую траекторию) системы. Кривая будет динамически перерисовываться.

Кстати, здесь мы использовали еще одну функцию `StreamDensityPlot`, изображающую линии тока на фоне графика плотности скалярного поля: в нашем случае на фоне поля нормы векторного поля.

Много других примеров представления решений ОДУ и их систем приведено в п. 4.5.

4.4 Дифференциально – алгебраические уравнения.

Система уравнений, связывающая неизвестные функции, может содержать алгебраические уравнения. Тогда ее называют системой дифференциально – алгебраических уравнений (система ДАУ).

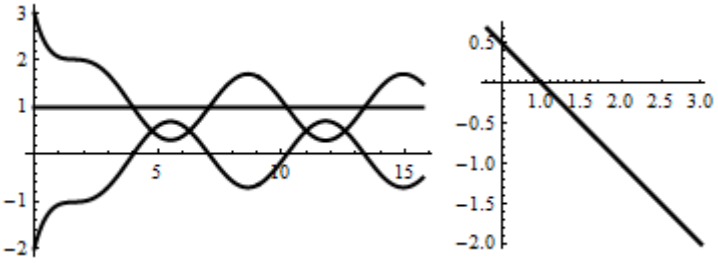
```
ds = DSolve[{x'[t] + y'[t] == x[t] + t^2, x[t] - y[t] == 1}, {x[t], y[t]}, t];
Simplify[ds[[1]]]
```

$\{x[t] \to -8 - 4t - t^2 + 2e^{t/2}C[1], y[t] \to -9 - 4t - t^2 + 2e^{t/2}C[1]\}$

Системы ДАУ могут содержать начальные условия

```
ds = DSolve[{x'[t] - y[t] == Sin[t], x[t] + y[t] == 1, x[0] == 3},
                                                {x[t], y[t]}, t];
Simplify[ds[[1]]]
Plot[Evaluate[{x[t], y[t], x[t] + y[t]}/. ds], {t, 0, 5π},
                                PlotStyle → {{Black, Thickness[0.01]}}]
ParametricPlot[Evaluate[{x[t], y[t]}/. ds], {t, 0, 5π},
                                  PlotStyle → {{Black, Thickness[0.02]}}]
```

$\{x[t] \to \frac{1}{2}(2 + 5e^{-t} - \text{Cos}[t] + \text{Sin}[t]), y[t] \to \frac{1}{2}(-5e^{-t} + \text{Cos}[t] - \text{Sin}[t])\}$



Смысл решенной здесь задачи состоит в определении параметрического уравнения алгебраической кривой, уравнение которой использовалось в системе ДАУ. В нашем случае это было уравнение прямой $x+y=1$. На правом графике мы построили эту прямую по найденным функциям $x(t)$, $y(t)$.

В системе ДАУ обязательно, чтобы хотя бы одно уравнение было дифференциальным. Вот пример системы двух ОДУ и одного алгебраического уравнения

```
DSolve[{x'[t] == x[t] + 2y[t], y'[t] == x[t] + z[t],
                                x[t] + y[t] + z[t] == 0}, {x[t], y[t], z[t]}, t]
```

$\{\{x[t] \to e^t C[1] + \frac{26}{27}e^{-t}C[2] + e^{-t}(-1 + e^{2t})C[2],$
$y[t] \to \frac{1}{27}e^{-t}C[2],$
$z[t] \to -e^t C[1] - e^{-t}C[2] - e^{-t}(-1 + e^{2t})C[2]\}\}$

Вот пример системы одного ОДУ и двух алгебраических уравнений

```
DSolve[{x'[t] == x[t] + y[t], y[t] == x[t] - 2z[t], x[t] + 2z[t] == 0},
                                                {x[t], y[t], z[t]}, t]
```

$\{\{x[t] \to -\frac{1}{8}e^{3t}C[1], y[t] \to -\frac{1}{4}e^{3t}C[1], z[t] \to \frac{1}{16}e^{3t}C[1]\}\}$

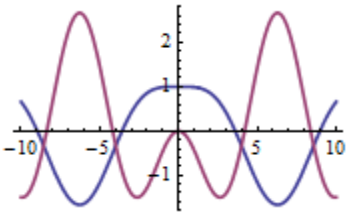
При постановке задачи важно правильно задавать начальные условия. Например, рассмотрим задачу $x(t)+y'(t)=0$, $y(t)=0$, $x(0)=1$, $y(0)=0$. Имеем

$y(t)=0 \Rightarrow y'(t)=0 \Rightarrow x(t)=0$. Т.о. решением является функции $x(t)=0$, $y(t)=0$. Но это несовместимо с условием $x(0)=1$.

Для системы ДАУ, содержащей ДУ 2-го порядка, надо задавать не менее 2 – х начальных условий

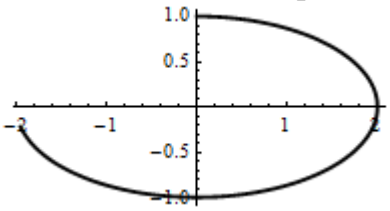
```
eqs = {x''[t] == 1/4 y[t], x[t] + y[t] == Cos[t], x[0] == 1, x'[0] == 0}
s = DSolve[eqs, {x[t], y[t]}, t]
Plot[{x[t]/. s[[1]], y[t]/. s[[1]]}, {t, -10, 10}, PlotStyle → Thickness[0.01]]
```

$\{\{x[t] \to \frac{1}{3}(4\text{Cos}[\frac{t}{2}] - \text{Cos}[t]), y[t] \to \frac{4}{3}(-\text{Cos}[\frac{t}{2}] + \text{Cos}[t])\}\}$



Символьное решение нелинейных систем ДАУ является трудной задачей и во многих случаях вместо `DSolve` можно использовать `NDSolve`.

```
s = NDSolve[{x'[t] == 2y[t] + x[t]y[t], x[t]^2/4 + y[t]^2 == 1, x[0] =
    = 0}, {x, y}, {t, 0, 10}]
ParametricPlot[Evaluate[{x[t], y[t]}/. s], {t, 0, 10}, PlotRange
    → All, AspectRatio → Automatic]
```



4.5 Примеры исследования ОДУ

В этом параграфе мы приводим примеры решения прикладных задач из области физики, механики, электротехники и других областей науки, которые сводятся к решению ОДУ или их систем.

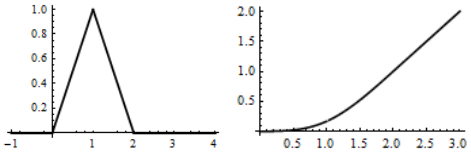
4.5.1 Движение материальной точки по прямой.

Дифференциальное уравнение движения имеет вид: $m\ddot{x} = f(t)$, где m – масса точки, $f(t)$ – внешняя сила.

Пример 1.1. Пусть внешняя сила на отрезке времени [*0,1*] задана в виде *f(t)=t*, на отрезке времени [*1,2*] равна *f(t)=2-t*, и равна 0 при *t>2*. Тогда решить задачу в *Mathematica* можно так (полагаем *m*=1).

```
f[t_] = Piecewise[{{0, t < 0}, {t, t < 1}, {2 - t, t < 2}}, 0];
Plot[f[t], {t, -1, 4}]
ds = DSolve[{x''[t] == f[t], x[0] == 0, x'[0] == 0}, x, t]
xs = x/. ds[[1]];
```

Plot[**xs**[t], {t, **0**, **3**}]



На левом рисунке приведен график внешней силы, а на правом – график функции $x(t)$ – решения ОДУ.

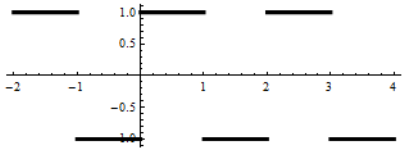
Пример 1.2. Внешняя сила является периодической импульсивной функцией, которая на разных участках периода принимает только два значения ±1. Определим такую функцию (*Periodic Impuls Function = Pif*)

$$Pif(x,a,w) = 2\cdot\left(\left[\frac{x}{w}\right]-\left[\frac{x-a}{w}\right]\right)-1$$

Это периодическая с периодом w функция. Для $a<w$ на отрезке $0 \le x < a$ функция равна 1, на остальном куске периода $a \le x < w$ функция равна -1 (случай $a \ge w$ мы не рассматриваем). На следующем рисунке приведен график функции *Pif(x,1,2)*.

Pif[x_, a_, w_] = **2**(**Floor**$\left[\frac{x}{w}\right]$ − **Floor**$\left[\frac{x-a}{w}\right]$) − **1**;

Plot[**Pif**[x, **1**, **2**], {x, −**2**, **4**}, **AspectRatio** → **Automatic**]

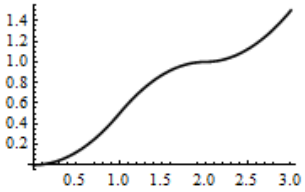


Пусть внешняя сила имеет вид *f(t)=Pif(x,1,2)* и *m=1*. Тогда

nds = **NDSolve**[{x''[t] == **Pif**[t, **1**, **2**], x[**0**] == **0**, x'[**0**] == **0**}, x, {t, **0**, **4**}]

xs = x/. **nds**[[**1**]];

Plot[**xs**[t], {t, **0**, **3**}]



4.5.2 Движение упругого мяча

Упругий мячик имеет начальное положение и скорость. Сопротивление воздуха пренебрежимо мало, но энергия движения расходуется при отскоке мяча от земли. Пусть при отскоке от земли вертикальная скорость мячика составляет 90% от вертикальной скорости в момент падения.

Уравнение движения (без учета сопротивления воздуха) имеет вид

$$m\ddot{\mathbf{r}} = -m\mathbf{g},$$

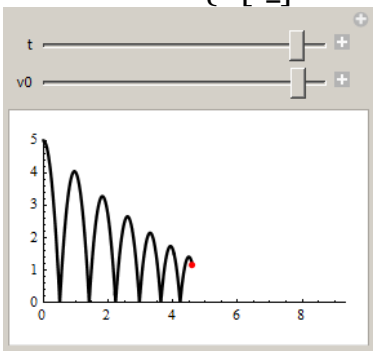
где $\mathbf{g}$ – вектор ускорения свободного падения, имеющий направление вертикально вниз. В покоординатной форме имеем $\ddot{x}=0$, $\ddot{y}=-g$ и начальные условия $x(0)=x_0$, $y(0)=h$, $\dot{x}(0)=v_0^x$, $\dot{y}(0)=v_0^y$. Векторное уравнение распадается на два независимых скалярных уравнения.

$$\begin{cases}\ddot{x}(t)=0\\ \ddot{y}(t)=-g\end{cases},$$

первое из которых имеет решение $x(t)=x_0+v_0^x\cdot t$. Это указывает на то, что в горизонтальном направлении движение происходит с постоянной скоростью v_0^x. В нашем примере положим $x_0=0$. Второе уравнение также интегрируется, если не учитывать скачкообразного изменения скорости в момент касания земли. Но мы хотим показать, как можно использовать функцию `WithEvent`, и будем решать уравнение численно. В момент отскока t_k вертикальная скорость $\dot{y}$ меняет знак, т.е. $\dot{y}_{после\ отскока}(t_k+\varepsilon)=-0.9\cdot\dot{y}_{до\ отскока}(t_k-\varepsilon)$ и становится положительной.

Находим решение второго уравнения системы с учетом резкого изменения скорости мяча при отскоке. Затем строим параметрическое уравнение траектории в манипуляторе с параметрами времени движения и начальной скорости.

```
Manipulate[
  s = NDSolveValue[{y''[t] == -9.81, y[0] == 5, y'[0] == 0,
        WhenEvent[y[t] == 0, y'[t] -> -0.9 y'[t]]}, y, {t, 0, 10}];
  p1 = ParametricPlot[Evaluate[{x[t], s[t]}], {t, 0, tmax},
                                  PlotRange → {{0, 10 v0}, {0, 5}}];
  p2 = Graphics[{RGBColor[1, 0, 0], Disk[{x[tmax], s[tmax]}, 0.1]}];
  Show[p1, p2],
  {{tmax, 0.5, t}, 0.3, 10}, {{v0, 0.5, "v0"}, 0.1, 1},
  AutoAction → False, Initialization: > {x[t_]: = v0 t; }]
```



4.5.3 Равновесие струны под действием сосредоточенных сил.

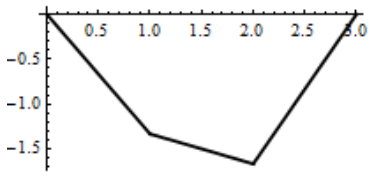
Пусть струна закреплена в точках $x_0=0$ и $x_n=L$, а внешние сосредоточенные силы f_i приложены в точках x_i: $0=x_0<x_1<x_2<\ldots<x_{n-1}<x_n=L$. Уравнение равновесия струны (с малым отклонением) имеет вид: $u''_{xx}=-f(x)$, $f(x)=\dfrac{F(x)}{T}$, где T – натяжение струны, $F(x)$ – внешняя сила, рассчитанная на единицу длины, $u(x)$ – отклонение струны от положения равновесия. Функцию $f(x)$, создаваемую набором сосредоточенных сил во внутренних точках x_i отрезка $[0, L]$, можно представить следующим образом $f(x)=\sum_{i=1}^{n-1} f_i\cdot\delta(x-x_i)$, где

$\delta(x-x_i)$ - функция Дирака, равная нулю везде, кроме точки x_i в которой она обращается в бесконечность так, что интеграл по отрезку, содержащему точку x_i, равен единице. В пакете *Mathematica* функция Дирака называется `DiracDelta[x]`. В примерах положим T=1.

Пример 3.1. Пусть $L=3,\ x_1=1,\ x_2=2,\ f_1=1,\ f_2=2$. Тогда

```
L = 3;
f[x_] = DiracDelta[x - 1] + 2DiracDelta[x - 2];
ds = DSolve[{u''[x] == f[x], u[0] == 0, u[L] == 0}, u, x]
z[x_] = Simplify[(ds[[1, 1, 2, 2]])]
Plot[z[x], {x, 0, L}]
```

$$-\frac{4x}{3}+2(-2+x)\text{HeavisideTheta}[-2+x]+(-1+x)\text{HeavisideTheta}[-1+x]$$



Мы изменили знак правой части специально, чтобы на графике прогиб струны был направлен вниз. Интересно отметить, что решение можно представить в более «приятном» виде: $u(x)=-\frac{5}{2}+\frac{x}{6}+\frac{1}{2}|x-1|+|x-2|$, который *Mathematica* не умеет получать с помощью функции `Simplify`. Для этого в полученном выражении для `z[x]` нужно сделать замену

```
(-k+x)HeavisideTheta[-k+x]->1/2(x-k-Abs[x-k]),
```

где `k` надо положить 1 и 2. Эта замена следует из формулы $x\,H(x)=\frac{1}{2}\left(x+|x|\right)$, где $H(x)$ – функция Хэвисайда (`HeavisideTheta`). Такая замена в более общем виде нам потребуется несколько раз, а именно $x^k\,H(x)=\frac{1}{2}\left(x^k+|x|^k\right)$ при нечетном k и $x^k\,H(x)=\frac{1}{2}\left(x^k+x^{k-1}|x|\right)$ при четном k. Напишем функцию, выполняющую такую замену

```
SimplifyHeavisideTheta = Function[{expr, xL, k, x},
   n = Length[xL];
   ch = Table[HeavisideTheta[-xL[[i]] + x], {i, n}];
   If[Mod[k, 2] == 0,
     cuh = Table[(-xL[[i]] + x)^k HeavisideTheta[-xL[[i]] + x] →
                 1/2((x - xL[[i]])^k + (x - xL[[i]])^(k-1) Abs[x - xL[[i]]]), {i, n}],
     cuh = Table[(-xL[[i]] + x)^k HeavisideTheta[-xL[[i]] + x] →
                 1/2((x - xL[[i]])^k + Abs[x - xL[[i]]]^k), {i, n}]
   ];
   cu = Simplify[Collect[expr, ch]];
   c2 = Simplify[(cu /. cuh)];
```

```
   cu2 = Simplify[Collect[c2, ch]];
   Simplify[(cu2/. cuh)]
];
```

Эта функция упрощает выражение `expr`, содержащее элементы вида $(-i+x)^k\, HeavisideTheta[-i+x]$, xL – список целых значений i, входящих в заменяемые выражения, k - целая положительная степень k=1,2,...; x - имя переменной в выражении `expr`.

Тогда, используя эту функцию, имеем

```
SimplifyHeavisideTheta[z[x], {1, 2}, 1, x]//TraditionalForm
```

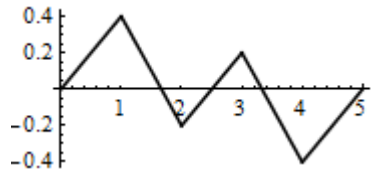
$\frac{1}{6}(6 \mid x-2 \mid +3 \mid x-1 \mid +x-15)$

Обратите внимание, что здесь решена краевая задача, а не задача Коши, как в предыдущем пункте.

Пример 3.2. Пусть L=5, сосредоточенные нагрузки f_{list} =[1,–1,1,–1] приложены в точках x_i =1, 2, 3, 4.

```
L = 5; xl = {1, 2, 3, 4}; fl = {1, −1, 1, −1};
f[x_] = Sum[fl[[i]]DiracDelta[x − xl[[i]]], {i, Length[xl]}];
nds = DSolve[{u''[x] == −f[x], u[0] == 0, u[L] == 0}, u, x];
z[x_] = (nds[[1, 1, 2, 2]]);
SimplifyHeavisideTheta[z[x], xl, 1, x]//TraditionalForm
Plot[z[x], {x, 0, L}, AspectRatio → 0.5]
```

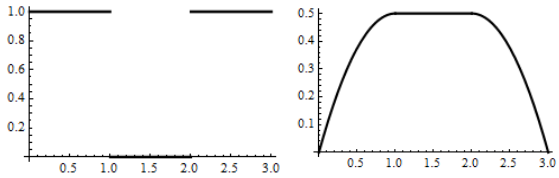
$\frac{1}{10}(5 \mid x-4 \mid -5 \mid x-3 \mid +5 \mid x-2 \mid -5 \mid x-1 \mid +4x-10)$



Пример 3.3. Равновесие струны под действием кусочно – постоянной силы.

```
L = 3;
f[x_] = HeavisideTheta[x] − HeavisideTheta[x − 1] +
                                        HeavisideTheta[x − 2];
ds = DSolve[{u''[x] == −f[x], u[0] == 0, u[L] == 0}, u, x];
SimplifyHeavisideTheta[ds[[1, 1, 2, 2]], {0, 1, 2}, 2, x]//TraditionalForm
GraphicsRow[{Plot[f[x], {x, 0, L}], Plot[nds[[1, 1, 2, 2]], {x, 0, L}]}]
```

$\frac{1}{4}(-x \mid x \mid -(x-2) \mid x-2 \mid +(x-1) \mid x-1 \mid -x^2+6x-3)$

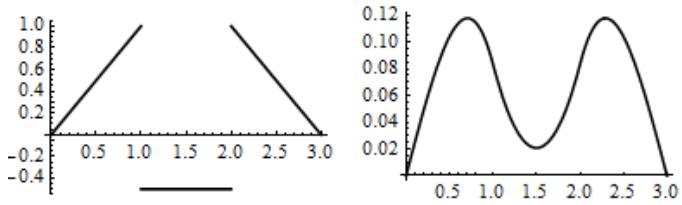


Слева показан график внешней силы, справа профиль струны.

Пример 3.4. Равновесие нити под действием кусочно - линейной внешней силы

```
L = 3; f[x_] = Piecewise[{{x, x < 1}, {−1/2, x < 2}}, 3 − x];
ds = DSolve[{u''[x] == −f[x], u[0] == 0, u[L] == 0}, u, x];
```

GraphicsRow[{**Plot**[$f[x]$, {x, **0**, L}], **Plot**[**ds**[[**1**, **1**, **2**, **2**]], {x, **0**, L}]}]



Слева показан график внешней силы, справа профиль нити. Заметим, что когда исходная функция f[x] задана как кусочная (Piecewise), то и результат получается в виде Piecewise функции. Если ту же функцию f[x] задать с использованием функции Хэвисайда, то результат можно будет привести к выражению, содержащему функции абсолютного значения.

$f[x\_] = x(1 - \mathbf{HeavisideTheta}[x - 1]) - (\mathbf{HeavisideTheta}[x - 1]$
$- \mathbf{HeavisideTheta}[x - 2])/2 + (3 - x)\mathbf{HeavisideTheta}[x - 2];$

$\mathbf{ds} = \mathbf{DSolve}[\{u''[x] == -f[x], u[0] == 0, u[L] == 0\}, u, x];$

$\mathbf{ss} = \mathbf{SimplifyHeavisideTheta}[\mathbf{ds}[[1, 1, 2, 2]], \{1, 2\}, 2, x]$

ss//TraditionalForm

$$\frac{1}{24}((x-2)(2x-13)|x-2| + (x-1)(2x+7)|x-1| - 9(2(x-3)x+5))$$

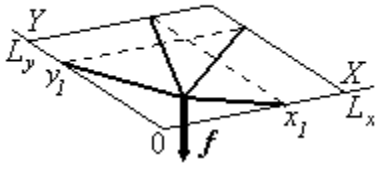
или

Collect[**Expand**[**ss**], {**Abs**[$-1 + x$], **Abs**[$-2 + x$]}]//**TraditionalForm**

$$\left(\frac{x^2}{12} - \frac{17x}{24} + \frac{13}{12}\right)|x-2| + \left(\frac{x^2}{12} + \frac{5x}{24} - \frac{7}{24}\right)|x-1| - \frac{3x^2}{4} + \frac{9x}{4} - \frac{15}{8}$$

4.5.4 Статический прогиб сети нитей

Пример 4.1 Рассмотрим механическую систему, состоящую из двух пересекающихся под прямым углом струн/нитей L_x и L_y длиной L. Нити в точке пересечения (x_1, y_1) соединены и в этой точке приложена сосредоточенная сила f_0. Под действием силы f_0 обе нити примут вид ломаной (см. следующий рисунок)



Требуется составить уравнения этих ломаных.

Очевидно, что действие силы f_0 частично уравновешивается струной L_x и частично струной L_y. Т.е. на каждую струну действует своя сосредоточенная сила. Это значит, что на нить L_x действует только часть силы f_0. Обозначим ее через a. Часть силы f_0, действующую на нить L_y, обозначим через b. Очевидно, что $a + b = f_0$. Если знать силы a и b, то можно, решив соответствующее ОДУ, определить прогиб каждой из струн.

Имея ввиду впоследствии рассмотреть сеть нитей, т.е. множество нитей с прогибами $u_1(x),\ldots$ и $v_1(y),\ldots$ обозначим функцию прогибов нити L_x через

$u[1][x]$, а нити L_y – через $v[1][y]$, а параметры через a[1] и b[1]. Составим и решим ОДУ прогиба первой и второй нитей при $x_1 = 1, y_1 = 2$

```
L = 3; f0 = 3;
dsX = DSolve[{u[1]''[x] == -a[1]DiracDelta[x - 1],
              u[1][0] == 0, u[1][L] == 0}, {u[1][x]}, x];
uu[x_] = SimplifyHeavisideTheta[dsX[[1, 1, 2]], {1}, 1, x]
dsY = DSolve[{v[1]''[y] == -b[1]DiracDelta[y - 2],
              v[1][0] == 0, v[1][L] == 0}, {v[1][y]}, y];
vv[y_] = SimplifyHeavisideTheta[dsY[[1, 1, 2]], {2}, 1, y]
```

$$\frac{1}{6}a[1](3 + x - 3\text{Abs}[-1 + x])$$

$$-\frac{1}{6}(-6 + y + 3\text{Abs}[-2 + y])b[1]$$

В решении присутствуют неопределенные пока параметры $a[1]$ и $b[1]$. Но у нас есть еще два условия: прогиб нитей в точке пересечения одинаков, и сумма этих параметров равна приложенной в узле (x_1, y_1) силе f0. Составляем алгебраическую систему уравнений и решаем ее

```
ss = Solve[{uu[1] == vv[2], a[1] + b[1] == f0}, {a[1], b[1]}]
```

$$\{\{a[1] \to \frac{3}{2}, b[1] \to \frac{3}{2}\}\}$$

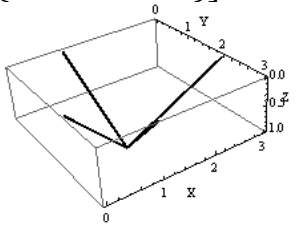
Зная параметры $a[1]$ и $b[1]$, построим функции прогибов нитей и нарисуем их графики

```
U1[x_] = uu[x]/.ss[[1]]
V1[y_] = vv[y]/.ss[[1]]
```

$$\frac{1}{4}(3 + x - 3\text{Abs}[-1 + x])$$

$$\frac{1}{4}(6 - y - 3\text{Abs}[-2 + y])$$

```
su1 = ParametricPlot3D[{t, 2, U1[t]}, {t, 0, L}];
sv1 = ParametricPlot3D[{1, t, V1[t]}, {t, 0, L}];
Show[su1, sv1, AxesLabel → {"X", "Y", "Z"}]
```



Пример 4.2 Рассмотрим сеть, составленную из двух пар пересекающихся нитей/струн одинаковой длины и одинакового натяжения, имеющие симметричное расположение в плоскости гипотетической квадратной мембраны размера L. Пусть струны в точках пересечения соединены и в узлах приложены одинаковые вертикальные сосредоточенные силы f. В каждом узле (i, j) такая сила уравновешивается частично натяжением нити одного семейства и частично натяжением нити второго семейства. Можно сказать, что на эти нити действуют по отдельности сосредоточенные силы, сумма которых равна

внешней силе приложенной в этом узле. Пусть уравнение ломаных первого семейства будут $u_1(x)$ и $u_2(x)$, а уравнения ломаных второго семейства $v_1(y)$ и $v_2(y)$. Обозначим узлы сети через (i, j), где $i, j=1,2$, сосредоточенные силы, приложенные к нитям первого семейства, через a_{ij}, и сосредоточенные силы, приложенные к нитям второго семейства – через b_{ij}. Очевидно, что $a_{ij}+b_{ij}=f$ для любых i и j. Рассматривая a_{ij} и b_{ij} как параметры, составим и решим соответствующие ОДУ прогиба нитей

```
Remove[a, b, u, v, UU, VV, U, V]
L = 3; XL = Table[i, {i, L - 1}]; YL = Table[j, {j, L - 1}];
m = Length[XL]; n = Length[YL];
f = Table[1, {i, m}, {j, n}];
dsX = DSolve[
        Flatten[Table[
            {u[j]''[x] == -Sum[a[j, k] DiracDelta[x - XL[[k]]], {k, m}],
                u[j][0] == 0, u[j][L] == 0}, {j, n}]],
          Table[u[j][x], {j, n}], x];
sx = SimplifyHeavisideTheta[FullSimplify[dsX[[1]]], XL, 1, x];
Table[UU[j][x_] = sx[[j, 2]], {j, n}];
dsY = DSolve[Flatten[Table[
            {v[i]''[y] == -Sum[b[i, k] DiracDelta[y - XL[[k]]], {k, n}],
                v[i][0] == 0, v[i][L] == 0}, {i, m}]],
            Table[v[i][y], {i, m}], y];
sy = SimplifyHeavisideTheta[FullSimplify[dsY[[1]]], YL, 1, y];
Table[VV[i][y_] = sy[[i, 2]], {i, m}];
```

Здесь функции UU[j][x] и VV[i][y] содержат неопределенные пока константы a[j,k] и b[i,k]. Составим систему линейных алгебраических уравнений относительно этих величин и решим ее.

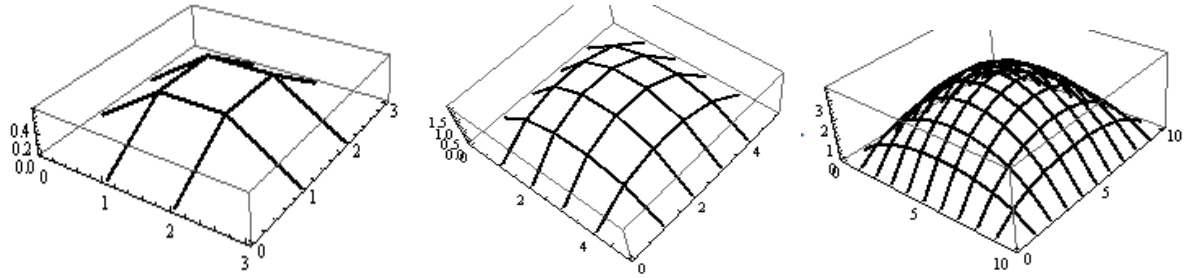
```
tt = Join[
      Flatten[Table[UU[j][XL[[i]]] == VV[i][YL[[j]]], {i, m}, {j, n}]],
      Flatten[Table[a[i, j] + b[j, i] == f[[i, j]], {i, m}, {j, n}]]];
tv = Flatten[Join[Table[a[i, j], {i, m}, {j, n}], Table[b[i, j], {i, m}, {j, n}]]];
ss = Solve[tt, tv];
```

$$\{\{a[1,1] \to \frac{1}{2}, a[1,2] \to \frac{1}{2}, a[2,1] \to \frac{1}{2}, a[2,2] \to \frac{1}{2},$$
$$b[1,1] \to \frac{1}{2}, b[1,2] \to \frac{1}{2}, b[2,1] \to \frac{1}{2}, b[2,2] \to \frac{1}{2}\}\}$$

Зная параметры a[j,k] и b[i,k], выполним их подстановку в функции UU[j][x] и VV[i][y], и построим кривые прогибов нитей U[j][x] и V[i][y].

```
Table[U[j][x_] = UU[j][x] /. ss[[1]], {j, n}];
Table[V[i][y_] = VV[i][y] /. ss[[1]], {i, m}];
p = Table[ParametricPlot3D[{t, YL[[j]], U[j][t]}, {t, 0, L},
                DisplayFunction → Identity], {j, n}];
q = Table[ParametricPlot3D[{XL[[i]], t, V[i][t]}, {t, 0, L},
                DisplayFunction → Identity], {i, m}];
```

Show[$\boldsymbol{p}, \boldsymbol{q}$]

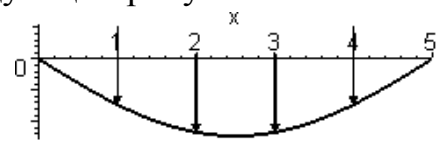


На предыдущем рисунке слева показана сеть, составленная из двух пар ортогонально пересекающихся упругих нитей. Если положить в предыдущем коде (сценарии) L=5, то получится сеть, показанная на предыдущем рисунке в середине. Положив L=10, мы получим деформированную сеть, показанную справа.

4.5.5 Прогиб балки

Уравнение изгибающего момента свободно опертой балки имеет тот же вид, что и уравнение равновесия струны $\frac{d^2M}{dx^2} = -q$, где *q(x)* внешняя сила, а *M(x)* изгибающий момент в балке, *x* – координата, направленная вдоль оси балки. Это значит, если балка находится под действием сосредоточенных или кусочных внешних сил, то мы можем применить прежний код для определения изгибающего момента *M*(*x*). Для определения прогиба балки нужно решать уравнение $E\,J\,\frac{d^2u(x)}{d\,x^2} = -M(x)$, *J(x)* – момент инерции поперечного сечения балки, *E* – модуль упругости материала балки, *u*(*x*) – отклонение нейтральной оси балки от положения равновесия. Также можно сразу решать ОДУ относительно функции прогиба $\frac{d^2}{dx^2}\left(EJ\,\frac{d^2u}{dx^2}\right) = q(x)$. Для свободно опертой на краях балки граничные условия должны иметь вид `u[0]=0,u[L]=0, u''[0]=0, u''[L]=0`. При ином закреплении граничные условия будут другими.

Пример 5.1 Пусть балка свободно оперта по краям и находится под действием четырех одинаковых сосредоточенных нагрузок. Нагрузки приложены на одинаковом расстоянии друг от друга и от концов балки. Решим модельную задачу, в которой положим *L*=5 (длина балки), *E*=8, *J*=1, *h*=1(высота балки). Единичные сосредоточенные нагрузки приложены в точках $x_i = 1, 2, 3, 4$. Схема нагрузок показана на следующем рисунке.

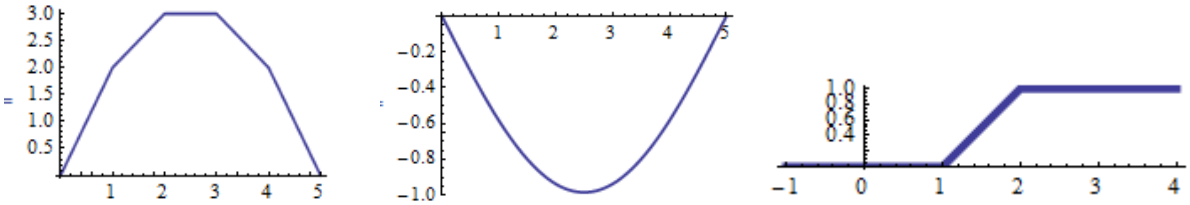


Найдем изгибающий момент *M*(*x*) балки. Его график показан на следующем рисунке слева.

Remove[$\boldsymbol{M}, \boldsymbol{f}, \boldsymbol{u}, \boldsymbol{x}, \boldsymbol{U}, \boldsymbol{W}$]; $\boldsymbol{L = 5}$;

```
f[x_] = Sum[DiracDelta[x - i], {i, 4}];
sM = DSolve[{M''[x] == -f[x], M[0] == 0, M[L] == 0}, M, x];
M = sM[[1, 1, 2]];
SimplifyHeavisideTheta[M[x], {1, 2, 3, 4}, 1, x]//TraditionalForm
Plot[M[x], {x, 0, L}, PlotStyle → Thickness[0.01]]
```

$$\frac{1}{2}\left(10-\left|x-1\right|-\left|x-2\right|-\left|x-3\right|-\left|x-4\right|\right)$$



Находим форму нейтральной оси балки.

```
Eu = 8; J = 1;
sU = DSolve[{Eu · J · u''[x] == M[x], u[0] == 0, u[L] == 0}, u, x];
W = sU[[1, 1, 2]];
ss = SimplifyHeavisideTheta[W[x], {1, 2, 3, 4}, 3, x];
U[x_] = Simplify[Expand[ss]]
U[x]//TraditionalForm
Plot[U[x], {x, 0, L}, PlotStyle → Thickness[0.01]]
```

$$\frac{1}{96}\left(100-150\,x+30\,x^2-\left|x-1\right|^3-\left|x-2\right|^3-\left|x-3\right|^3-\left|x-4\right|^3\right)$$

График функции $U(x)$ показан на предыдущем рисунке в середине.

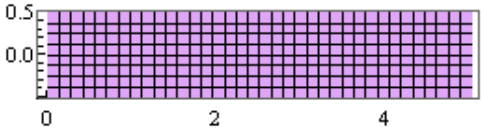
Целью следующего блока кода является графическое представление продольных напряжений в балке. Вначале напишем код, который рисует прямоугольную балку. Он использует вспомогательную функцию

```
P[x_, a_, w_] = (w + Abs[x - a] - Abs[x - a - w])/2;
Plot[P[x, 1, 1], {x, -1, 4}, AspectRatio → Automatic]
```

График функции $P(x,a,w)$ при $a=1, w=1$ показан на предыдущем рисунке справа.

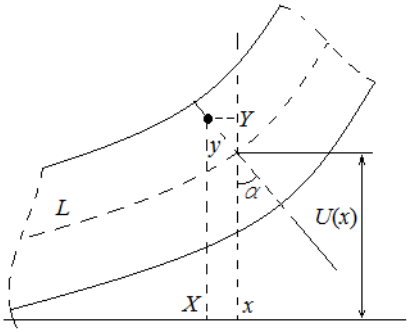
Теперь составляем параметрическое уравнение области прямоугольника, представляющего балку (с методикой составления параметрических уравнений ограниченных областей вы можете познакомиться по пособиям автора, представленным в разделах спецкурса «Аналитические методы геометрического моделирования»). Для графического представления области балки мы используем параметрическое уравнение ее поверхности в пространстве с третьей координатой Z=0.

```
h = 1; hh = h/2; (* полувысотабалки *)
x[u_, v_] = P[u, 0, L];
y[u_, v_] = -hh + P[v, -hh, 2hh];
ParametricPlot3D[{x[u, v], y[u, v], 0}, {u, 0, L}, {v, -hh, hh},
                 ViewPoint → {0, 0, Infinity}, Mesh → {35, 7}]
```



Используя уравнения плоской области $x(u,v)$, $y(u,v)$ недеформированной балки и уравнение $U(x)$ ее нейтральной оси, можно написать параметрическое уравнение области деформированной балки.

В соответствии с гипотезой плоских сечений, все поперечные сечения, которые были перпендикулярны нейтральной оси балки до деформирования, остаются перпендикулярными нейтральной оси балки после ее деформирования.



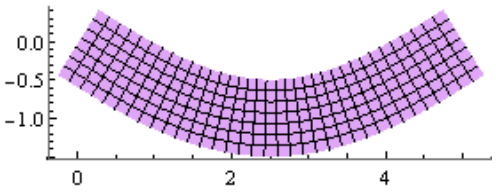
Точки, имевшие до деформирования координаты (x,y), переходят в точки с координатами (X,Y), где $X = x - y\cos\alpha$, $Y = U(x) + y\cos\alpha$ (см. рисунок). Но $tg\,\alpha = U'(x)$ и, поэтому, $X = x - \dfrac{U'_x(x)y}{\sqrt{1+(U'_x(x))^2}}$ и $Y = U(x) + \dfrac{y}{\sqrt{1+(U'_x(x))^2}}$. Тогда функции $X(u,v), Y(u,v)$, представляющие параметрические уравнения области изогнутой балки будут иметь вид

$$X(u,v) = x(u,v) - \frac{U'_x(x(u,v))y(u,v)}{\sqrt{1+(U'_x(x(u,v)))^2}}$$

$$Y(u,v) = U(x(u,v)) + \frac{y(u,v)}{\sqrt{1+(U'_x(x(u,v)))^2}}$$

Строим область деформированной балки. Для этого вычисляем производную $U_1(x) = U'_x(x)$ и создаем вспомогательную функцию $\sqrt{1+(U'_x(x))^2}$.

```
U1[x_] = Simplify[U'[x]/.Table[Abs[-i + x]^2 Abs'[-i + x] →
                    (-i + x) Abs[-i + x], {i, 4}], x ∈ Reals]
Usq[x_] = √(1 + U1[x]^2);
X[u_, v_] = x[u, v] - U1[x[u, v]] y[u, v] / Usq[x[u, v]];
Y[u_, v_] = U[x[u, v]] + y[u, v] / Usq[x[u, v]];
ParametricPlot3D[{X[u, v], Y[u, v], 0}, {u, 0, L}, {v, -hh, hh},
                 ViewPoint → {0, 0, Infinity}, Mesh → {34, 6}]
```



Следующим шагом будет показать в цвете продольные напряжения σ_x в балке. Если поперечное сечение балки прямоугольное, то σ_x вычисляются по формуле $\sigma_x(x,y)=\frac{M(x)\cdot y}{J}$, где $M(x)$ – момент инерции в сечении x балки, y – вертикальное смещение точки балки относительно нейтральной оси. Чтобы использовать эту функцию в качестве функции цвета, ее нужно привести к диапазону [0, 1]. Для этого нужно знать минимальные и максимальные значения напряжения σ_x. В нашей задаче они, очевидно, равны напряжениям в срединном сечении балки в верхней и нижней точках $\sigma_{x\,\max/\min}=\pm\frac{M(L/2)\cdot(h/2)}{J}$, где h – высота балки.

Создаем функцию продольных напряжений tensionX и нормированную в диапазоне от 0 до1 функцию beamColor.

$$\mathbf{tensionX[u\_, v\_] = M[x[u, v]] \cdot y[u, v]/J;}$$

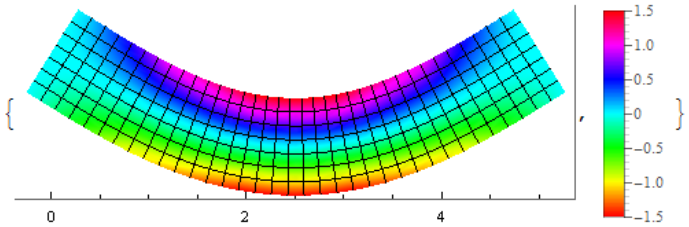
$$\mathbf{tensmin = -\frac{M[L/2]hh}{J};}$$

$$\mathbf{tensmax = \frac{M[L/2]hh}{J};}$$

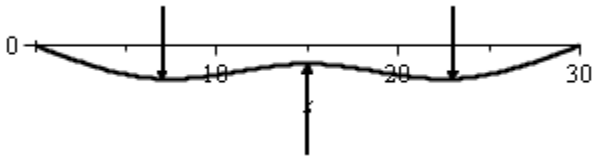
$$\mathbf{beamColor[u\_, v\_] = \frac{tensionX[u, v] - tensmin}{tensmax - tensmin};}$$

Рисуем область изогнутой балки, раскрашенную в соответствии с продольными напряжениями. Рядом рисуем палитру напряжений.

```
{ParametricPlot3D[{X[u, v], Y[u, v], 0}, {u, 0, L}, {v, -hh, hh},
    ViewPoint → {0, 0, Infinity}, Mesh → {34, 6},
    ColorFunction → (Hue[beamColor[#4, #5]]&),
    ColorFunctionScaling → False, Axes → {True, False, False}],
 BarLegend[{Hue, {-1.5, 1.5}}, LegendMarkerSize → 150]}
```



Пример 5.2 Пусть балка свободно оперта по краям и находится под действием трех сосредоточенных нагрузок. Схема приложения нагрузок показана на следующем рисунке.



Решим задачу, в которой положим L=30, E=2, J=54, h=6. Сосредоточенные нагрузки $f_i = 4, -5, 4$ приложены в точках $x_i = 7, 15, 23$. Приведем только строки кода, которые изменены по сравнению с предыдущей задачей

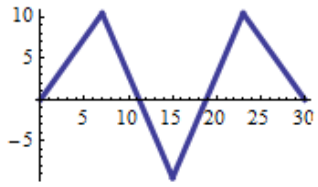
$$\mathbf{Remove}[M, f, u, x, U, W]; L = 30;$$

$$f[\mathrm{x\_}] = \mathbf{4DiracDelta}[x - 7] - \mathbf{5DiracDelta}[x - 15] + \mathbf{4DiracDelta}[x - 23]$$

…

$$\mathbf{SimplifyHeavisideTheta}[M[x], \{7, 15, 23\}, 1, x]//\mathbf{TraditionalForm}$$

…



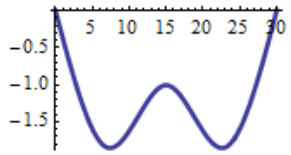
$$\mathrm{M[x]} = \frac{1}{2}(-4 \mid x - 23 \mid + 5 \mid x - 15 \mid - 4 \mid x - 7 \mid + 45)$$

$$\mathbf{Eu} = 2; J = 54;$$

…

$$\mathbf{ss} = \mathbf{SimplifyHeavisideTheta}[W[x], \{7, 15, 23\}, 3, x];$$

…



$$\frac{-4|x-23|^3 + 5|x-15|^3 - 4|x-7|^3 + 135x^2 - 4050x + 33165}{1296}$$

$$h = 6; \mathbf{hh} = h/2;$$

…

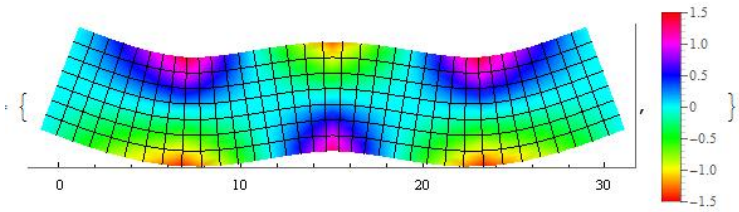
$$\mathbf{U1}[\mathrm{x\_}] = \mathbf{Simplify}[U'[x]/.\{\mathbf{Abs}[-7 + x]^2\mathbf{Abs}'[-7 + x] \to (-7 + x)\mathbf{Abs}[-7 + x],$$
$$\mathbf{Abs}[-15 + x]^2\mathbf{Abs}'[-15 + x] \to (-15 + x)\mathbf{Abs}[-15 + x],$$
$$\mathbf{Abs}[-23 + x]^2\mathbf{Abs}'[-23 + x] \to (-23 + x)\mathbf{Abs}[-23 + x]\}, x \in \mathbf{Reals}]$$

…

$$\mathbf{tensmin} = -\frac{M[7]\mathbf{hh}}{J};$$

$$\mathbf{tensmax} = \frac{M[7]\mathbf{hh}}{J};$$

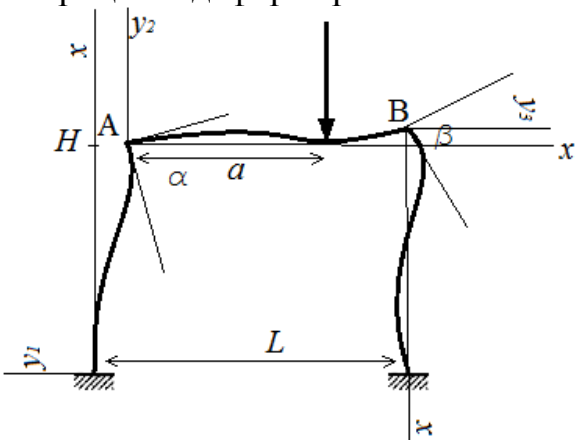
…



Для экономии места мы также не привели промежуточные рисунки, которые генерирует наш код.

4.5.6 Расчет рамы

Жесткой рамой называется такая стержневая система у которой все узловые соединения являются жесткими. Жесткий узел характеризуется тем, что угол между осями стержней его образующих не изменяется при действии нагрузки. Такой пример показан на следующем рисунке – углы α и β между касательными к осям горизонтальной и вертикальным балкам остается неизменным (прямым) в процессе деформирования.

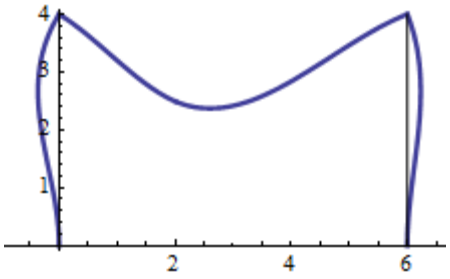


Когда рама деформируется, то обе касательные, проведенные в узле А, поворачиваются на одинаковый угол, иначе величина угла α изменилась бы. Тоже самое можно сказать и о касательных в точках В. С математической точки зрения расчет рамы сводится к решению системы ОДУ, описывающих деформацию стержней, задания граничных условий закрепления и сопряжения.

Рассмотрим раму, показанную на предыдущем рисунке. Размеры таковы: L=6, H=4, a=2. Положим также, что вертикальные и горизонтальные смещения узлов А и В равны нулю (они малы и ими можно пренебречь). Свяжем с каждым стержнем свою систему координат $(x, y_i)(i=1,2,3)$, направления осей которых показаны на рисунке. Прогиб левого вертикального стержня обозначим через $y_1(x)(0 \le x \le H)$, прогиб верхнего горизонтального обозначим через $y_2(x)(0 \le x \le L)$, прогиб правого вертикального стержня – через $y_3(x)(0 \le x \le H)$. Условия неизменности углов (жесткие узлы А и В) будут выполнены, если в этих точках совпадают производные к соответствующим прогибам. Т.е. эти условия имеют вид $y'_1(H) = y'_2(0), y'_2(L) = y'_3(0)$. Также в этих узлах одинаковые изгибающие моменты сопрягаемых стержней $E_1 J_1 \frac{d^2 y_1(H)}{dx^2} = E_2 J_2 \frac{d^2 y_2(0)}{dx^2}$ и $E_2 J_2 \frac{d^2 y_2(L)}{d\,x^2} = E_3 J_3 \frac{d^2 y_3(0)}{dx^2}$. Условия жесткого защемления опор дают граничные условия вида $y_1(0) = 0, y'_1(0) = 0$, $y_3(H) = 0, y'_3(H) = 0$. Тогда для решения задачи имеем следующий код

```
H = 4; L = 6; E1 = 1; J1 = 1; E2 = 1; J2 = 1; E3 = 1; J3 = 1;
ss = DSolve[{
        E1J1u1''''[x] == 0,
```

```
    E2J2u2''''[x] == -DiracDelta[x - 2],
    E3J3u3''''[x] == 0,
    u1[0] == 0, u1[H] == 0, u1'[0] == 0, E1J1u1''[H] == E2J2u2''[0],
    u1'[H] == u2'[0], u2[0] == 0, u2[L] == 0, u3[0] == 0,
    E2J2u2''[L] == E3J3u3''[0], u2'[L] == u3'[0], u3[H] == 0,
    u3'[H] == 0}, {u1, u2, u3}, x]
y1 = u1/. ss[[1]];
y2 = u2/. ss[[1]];
y3 = u3/. ss[[1]];
p1 = ParametricPlot[{-y1[t], t}, {t, 0, H}];
p2 = ParametricPlot[{t, H + y2[t]}, {t, 0, L}];
p3 = ParametricPlot[{L + y3[t], H - t}, {t, 0, H}];
p4 = Graphics[Line[{{L, 0}, {L, H}}]];
Show[{p1, p2, p3, p4}, PlotRange → {{-1, L + 1}, {-1, H + 1}}]
```



Для представления формы деформированной рамы, мы записали параметрические уравнения кривых(осей стержней) в глобальной системе координат. Выражения для прогибов в локальных координатах имеют вид

```
y1[x]
SimplifyHeavisideTheta[y2[x], {2}, 3, x] //TraditionalForm
y3[x]
```

$$-\frac{11}{288}(-4x^2+x^3)$$

$$\frac{1}{324}(-27\,|x-2|^3+11x^3+63x^2-522x+216)$$

$$\frac{7}{288}(16x-8x^2+x^3)$$

4.5.7 Поперечные колебания струны с грузами

Пример 7.1 *Поперечные колебания струны с двумя грузами.* Имеется 2 груза массы *M*, расположенные на струне длиной $L=3\,a$ в точках $x=a$ и $x=2a$. Отрезки струны между грузами одинаковы, невесомы и подчиняются закону Гука. Натяжение в равновесии равно T. Обозначим вертикальное отклонение грузов от положения равновесия на оси струны через $y_1(t)$ и $y_2(t)$. Ограничимся рассмотрением малых колебаний. Уравнения движения грузов имеют вид

$$M\frac{d^2y_1}{dt^2}=T\left(\frac{y_2-y_1}{a}\right)-T\left(\frac{y_1-y_0}{a}\right)$$

$$M\frac{d^2y_2}{dt^2}=T\left(\frac{y_3-y_2}{a}\right)-T\left(\frac{y_2-y_1}{a}\right)$$

где y_0, y_3 - смещения левого и правого концов закрепленной струны, т.е. нули. Положим a=1, M=1, T=1. Тогда получим систему

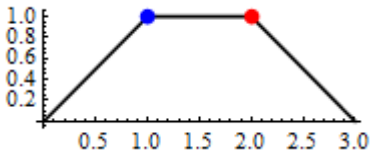
$$\begin{cases}\dfrac{d^2y_1}{dt^2}=y_2-2y_1\\ \dfrac{d^2y_2}{dt^2}=-2y_2+y_1\end{cases}$$

Рассмотрим колебание системы без начальной скорости. Для этого решим систему ОДУ относительно функций $y_1(t), y_2(t)$ с единичными начальными смещениями грузов и нулевыми начальными скоростями.

```
Remove[y1, y2]
sys = {y1''[t] == y2[t] - 2y1[t], y2''[t] == -2y2[t] + y1[t],
       y1[0] == 1, y2[0] == 1, y1'[0] == 0, y2'[0] == 0}
ds = DSolve[sys, {y1, y2}, t]
y1 = ds[[1, 1, 2]]
y2 = ds[[1, 2, 2]]
```

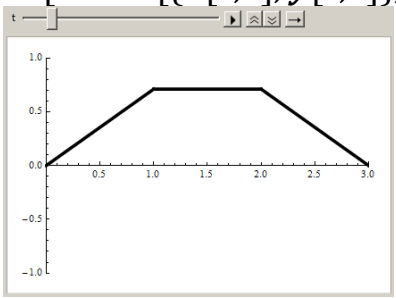
Зная функции $y_1(t), y_2(t)$, составим кусочное параметрическое уравнение струны и нарисуем начальное состояние системы

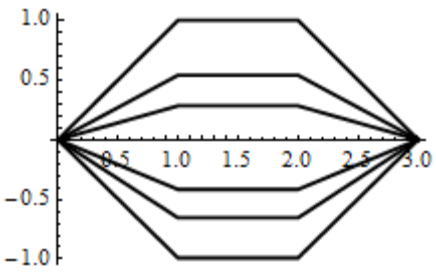
```
x[τ_, t_] = τ;
y[τ_, t_] = Piecewise[{{y1[t]τ, τ < 1},
                       {y1[t] + (y2[t] - y1[t])(τ - 1), τ < 2},
                       {y2[t] - y2[t](τ - 2), τ < 3}}, 0]
ParametricPlot[{x[t, 0], y[t, 0]}, {t, 0, 3},
 pilog-> {PointSize[Large], Blue, Point[{1, y1[0]}], Red, Point[{2, y2[0]}]}]
```



Теперь построим анимацию колебаний системы (след. рисунок слева) и график нескольких положений струны в различные моменты времени (рисунок справа).

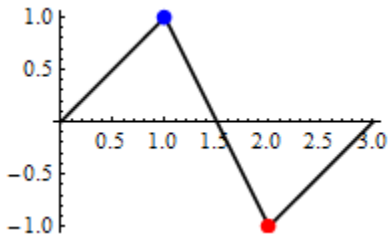
```
Animate[
   ParametricPlot[Evaluate[{x[τ, t], y[τ, t]}], {τ, 0, 3},
                  PlotRange → {{0, 3}, {-1, 1}}],
   {t, 0, 6}, AnimationRunning → False]
ParametricPlot[Table[{x[t, i], y[t, i]}, {i, 0, 5}], {t, 0, 3}]
```

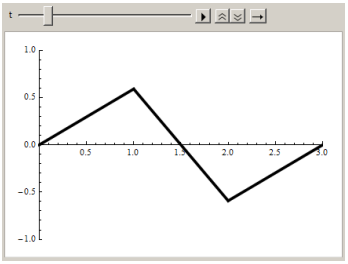




Полученное синхронное колебание обеих масс представляет первую моду колебаний системы. Для построения второй моды колебаний, представляющей движение масс в «противофазе», можно повторить весь предыдущий код, только надо задать начальные смещения тоже в «противофазе».

```
Remove[y1, y2]
sys = {y1''[t] == y2[t] - 2y1[t], y2''[t] == -2y2[t] + y1[t],
                y1[0] == 1, y2[0] == -1, y1'[0] == 0, y2'[0] == 0}
ds = DSolve[sys, {y1, y2}, t]
y1 = ds[[1, 1, 2]]
y2 = ds[[1, 2, 2]]
x[τ_, t_] = τ;
y[τ_, t_] = Piecewise[{{y1[t]τ, τ < 1},
                       {y1[t] + (y2[t] - y1[t])(τ - 1), τ < 2},
                       {y2[t] - y2[t](τ - 2), τ < 3}}, 0]
ParametricPlot[{x[t, 0], y[t, 0]}, {t, 0, 3},
 pilog-> {PointSize[Large], Blue, Point[{1, y1[0]}], Red, Point[{2, y2[0]}]}]
Animate[
    ParametricPlot[Evaluate[{x[τ, t], y[τ, t]}], {τ, 0, 3},
                   PlotRange → {{0, 3}, {-1, 1}}],
    {t, 0, 6}, AnimationRunning → False]
```





Начальное состояние системы показано на предыдущем рисунке слева, а панель анимации – справа. Для создания произвольных колебаний (суперпозиции первой и второй мод) достаточно задать произвольные начальные смещения. Попробуйте, например, задать условия $y_1(0)=1,\ y_2(0)=0$.

Пример 7.2 *Поперечные колебания струны с n грузами.* Имеется *n* грузов массы *M*, расположенных в точках $x=a, 2a, 3a,\ldots,na$. Отрезки невесомой струны/нити между грузами одинаковы и подчиняются закону Гука. Натяжение в равновесии равно T. Обозначим вертикальное отклонение грузов от положения равновесия на оси струны через $y_i(t)$ $(i=1,2,\ldots,n)$. Ограничимся рассмотрением малых колебаний. Уравнения движения имеют вид

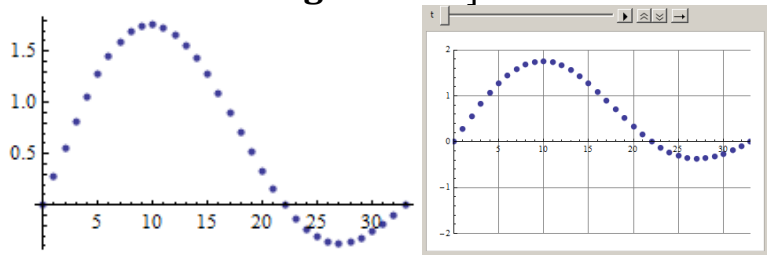
$$M\frac{d^2y_i}{dt^2}=T\left(\frac{y_{i+1}-y_i}{a}\right)-T\left(\frac{y_i-y_{i-1}}{a}\right)\quad (i=1,2,\ldots,n)$$

При этом, в силу условия закрепления струны на концах, следует считать, что $y_0(t)=0,\ y_{n+1}(t)=0$. Рассмотрим колебание системы без начальной скорости. Положим *a*=1, *M*=1, *T*=1. Тогда имеем

$$\frac{d^2 y_i}{dt^2} = y_{i+1} - 2y_i + y_{i-1} \quad (i = 1,2,\ldots,n)$$

Напишем код для решения этой системы ОДУ и построим анимацию движения грузов.

```
n = 32;
eqns = Flatten[Table[{y_i''[t] == (y_{i+1}[t] - 2y_i[t] + y_{i-1}[t]),
                      y_i[0] == Sin[πi/(n+1)] + Sin[2πi/(n+1)], y_i'[0] == 0}, {i, n}]];
eqns = Join[eqns, {y_0[t] == 0, y_{n+1}[t] == 0}];
vars = Table[y_i, {i, 0, n + 1}];
sol = NDSolve[eqns, vars, {t, 0, 2(n + 1)}];
ListPlot[Table[{i, y_i[0]}/. sol[[1]], {i, 0, n + 1}], PlotStyle → PointSize[0.02]]
Animate[
    ListPlot[Table[{i, y_i[t]}/. sol[[1]], {i, 0, n + 1}],
        PlotStyle → PointSize[0.02], PlotRange → {{0, n + 1}, {-2, 2}},
        GridLines → Automatic],
    {t, 0, 2 n}, AnimationRunning → False]
```

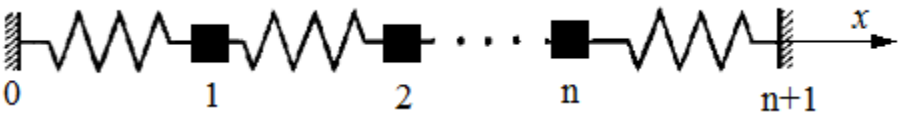


Здесь на левом графике показано начальное состояние системы, справа –панель анимации.

Попробуйте использовать большее/меньшее n. Задайте другие начальные смещения, например, $y_i[0] == \frac{n}{2} - \text{Abs}[i - \frac{n}{2}]$.

4.5.8 Модель Ферми – Улама – Пасты

Представим простейшую одномерную модель кристалла – цепочку одинаковых грузиков массы m, соединенных одинаковыми пружинками жесткостью k. Обозначим через y_i отклонение (вдоль оси X) грузика i от положения равновесия.



Тогда, в соответствии с законом Гука, уравнения движения i-го грузика имеет вид

$$m\frac{d^2 y_i}{dt^2} = k(y_{i+1} - y_i) - k(y_i - y_{i-1}) \quad (i = 1,2,\ldots,n),$$

который полностью совпадает с уравнениям поперечных колебаний невесомой струны с грузами.

Ферми с коллегами С. Уламом и Дж. Пастой предположил, что сила $F(y_{i+1}, y_i)$, с которой взаимодействуют (i+1) –й и i – й атомы имеет небольшую нелинейную добавку. Это значит, что возвращающая сила имеет вид $F(y_{i+1}, y_i) = k\,\Delta l + \alpha\,\Delta l^2$, где $\Delta l = y_{i+1} - y_i$ и α мало. Это приводит к следующей системе ОДУ

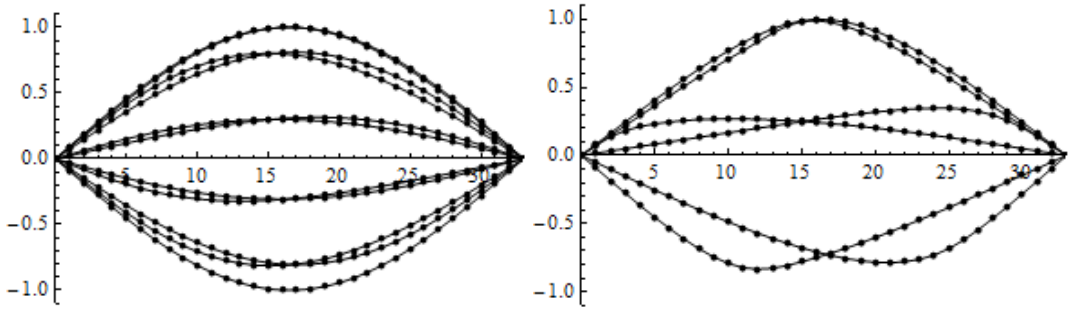
$$m\frac{d^2 y_i}{dt^2} = k(y_{i+1} - 2y_i + y_{i-1}) + \alpha\left((y_{i+1} - y_i)^2 - (y_i - y_{i-1})^2\right) \quad (i = 1,2,\ldots,n)$$

При решении этой системы Ферми с коллегами следили не за отдельными частицами, а за поведением мод колебаний, которые при $\alpha = 0$ совпадают с синусоидами·

Рассмотрим движение, при котором в начальный момент времени возбуждена только одна первая мода с периодом T. Код решения этой задачи повторяет код предыдущего примера с той разницей, что уравнения системы ОДУ теперь нелинейные. Положим k=1, n=32, α=1 и начальные смещения зададим в форме первой моды $y_i(0) = \sin\left(\frac{\pi i}{n+1}\right)$.

```
n = 32; a = 1;
T = 66; (* период линейных колебаний *)
eqns1 = Flatten[Table[
                {y_i''[t] == (y_(i+1)[t] - 2y_i[t] + y_(i-1)[t]) +
                      a · ((y_(i+1)[t] - y_i[t])^2 - (y_i[t] - y_(i-1)[t])^2),
                  y_i[0] == Sin[πi/(n+1)], y_i'[0] == 0}, {i, n}]];
eqns1 = Join[eqns1, {y_0[t] == 0, y_(n+1)[t] == 0}];
vars1 = Table[y_i, {i, 0, n + 1}];
sol1 = NDSolve[eqns1, vars1, {t, 0, 200(n + 1)}, MaxSteps → 100000];
p1 = Table[ListPlot[Table[
                {i, y_i[tk]}/. sol1[[1]], {i, 0, n + 1}],
                PlotStyle → {PointSize[0.01], Red}, PlotRange → All,
                Joined → True, Mesh → All],
        {tk, 0, T, T/10}];
Show[p1, PlotRange → {{0, n + 1}, {-1.1, 1.1}}]
```

Обратите внимание, что при решении системы ОДУ использована опция MaxSteps→100000, которая необходима для большого временного интервала, на котором определяется численное решение.



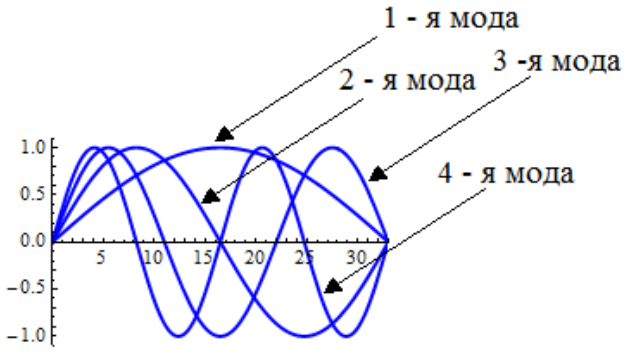
На предыдущем рисунке слева показаны формы колебаний в моменты времени $t = 0, \frac{T}{10}, \frac{2T}{10}, \ldots, T$. Следующий код строит графики решения при $t = 2T, 2T + \frac{T}{5}, \ldots, 3T$, которые показаны на предыдущем рисунке справа. Для наглядности на обоих графиках смещение грузов отложены в вертикальном направлении.

```
p2 = Table[ListPlot[Table[
                {i, yi[tk]}/. sol1[[1]], {i, 0, n + 1}],
                PlotStyle → {PointSize[0.01], Red}, PlotRange → All,
                Joined → True, Mesh → All],
        {tk, 2T, 3T, T/5}];
Show[p2, PlotRange → {{0, n + 1}, {−1.1, 1.1}}]
```

Как видим, форма решения все еще похожа на половину волны синусоиды, т.е. на первую моду колебаний. Однако, уже наблюдается сдвиг максимума в стороны от середины отрезка [0, n+1]. Первая мода еще узнаваема при $t \approx 7T$.

Следующий код строит графики 4 - х первых мод линейных колебаний.

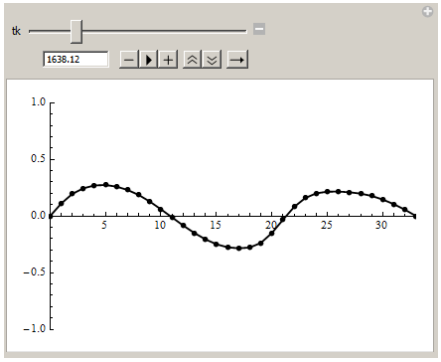
```
pl = Table[Plot[Sin[πxk/(n + 1)], {x, 0, n + 1},
                PlotStyle → {Blue, Thickness[0.01]}], {k, 1, 4}];
Show[pl, PlotRange → {{0, n + 1}, {−1.1, 1.1}}]
```

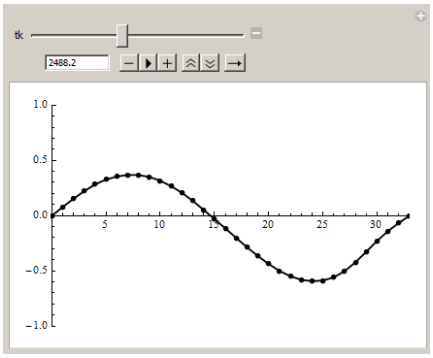


Вернемся к нелинейным колебаниям. При $t \approx 25T$ мы наблюдаем в основном 3 – ю моду. Чтобы в этом убедиться, достаточно использовать манипулятор (следующий рисунок слева) и код, приведенный ниже.

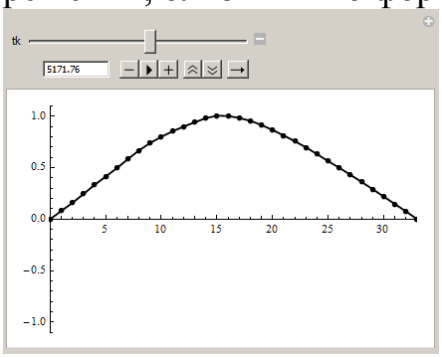
```
Manipulate[
    ListPlot[Table[{i, yi[tk]}/. sol1[[1]], {i, 0, n + 1}],
        PlotStyle → {PointSize[0.015], Black, Thickness[0.005]},
        PlotRange → {{0, n + 1}, {−1, 1}}, Joined → True, Mesh → All],
{tk, 24T, 28T, 0.01T}]
```

При $t \approx 38.5T$ мы наблюдаем в основном 2 – ю моду. Измените в предыдущем коде последнюю строку на $\{\text{tk}, 36T, 40T, 0.01T\}]$ и вы получите манипулятор, показанный справа.





Графики решений, построенные при $t \approx 51.5T$ снова напоминают 3 – ю моду. При $t \approx 77T$ мы снова возвращаемся к первой моде. Внесите изменение в последнюю строку предыдущего кода $\{\text{tk}, 75T, 80T, 0.01T\}]$ и вы получите манипулятор с графиками решений, близкими по форме к 1 – й моде.



Подведем итог, который впервые наблюдали Ферми и его сотрудники. В начальные моменты времени форма колебаний близка к форме первой моды, но потом начинается перекачивание энергии в другие моды, и форма решения больше напоминает сумму нескольких синусоид с различным периодом (сумма мод). При $t \approx 25T$ возбуждена в основном 3 – я мода, при $t \approx 38.5T$ преобладает 2 – я мода. Затем при $t \approx 51.5T$ форма опять близка к 3 – й моде, и при $t \approx 77T$ снова возвращается к первой моде.

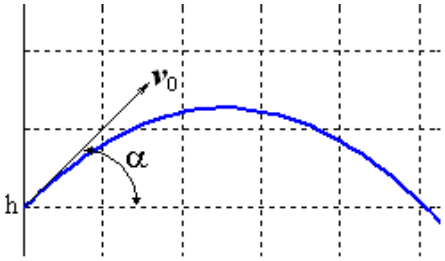
Численные эксперименты сотрудников Ферми показали, что такое поведение не случайность. Увеличение числа грузиков, изменение значения параметра α, изменение формы нелинейности (например, $\alpha \Delta l^3$ вместо $\alpha \Delta l^2$) не меняет поведение системы. Моды не сливаются, а происходит выделение некоторых из них, которые доминируют по очереди. Когда возвращается начальная мода, все повторяется. Время возвращения T_R (в нашем примере $T_R \approx 77\,T$) зависит от n, от вида нелинейности, но «солирование» низших мод и возвращение при $t = T_R$ наблюдалось сотрудниками Ферми во всех расчетах.

4.5.9 Движение тела, брошенного под углом к горизонту.

Решим задачу Коши, описывающую движение тела, брошенного с начальной скоростью v_0 под углом α к горизонту в предположении, что сопротивление воздуха пропорционально квадрату скорости. В векторной форме уравнение движения имеет вид

$$m\ddot{\mathbf{r}} = -\gamma \cdot \mathbf{v} \left| \mathbf{v} \right| - m\mathbf{g},$$

где $\mathbf{r}(t)$ радиус – вектор движущегося тела, $\mathbf{v} = \dot{\mathbf{r}}(t)$ – вектор скорости тела, γ – коэффициент сопротивления, $m\mathbf{g}$ вектор силы веса тела массы m, $\mathbf{g}$ – вектор ускорения свободного падения.



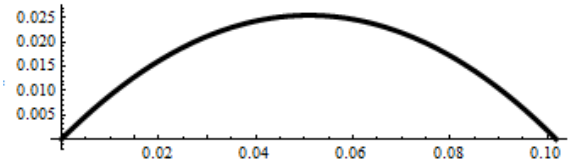
Особенность этой задачи состоит в том, что движение заканчивается в заранее неизвестный момент времени, когда тело падает на землю.

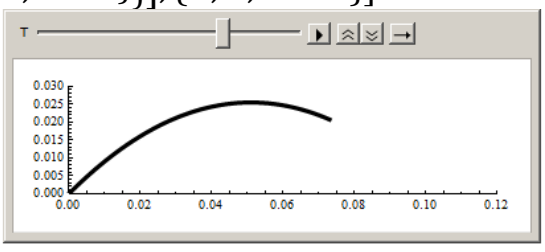
Если обозначить $k = \gamma / m$, то в координатной форме мы имеем систему уравнений

$$\ddot{x} = -k\,\dot{x}\sqrt{\dot{x}^2 + \dot{y}^2}$$
$$\ddot{y} = -k\,\dot{y}\sqrt{\dot{x}^2 + \dot{y}^2} - g$$

к которой следует добавить начальные условия: $x(0) = 0$, $y(0) = h$ (h начальная высота), $\dot{x}(0) = v_0 \cos\alpha$, $\dot{y}(0) = v_0 \sin\alpha$.

```
α = π/4; v0 = 1; h = 0; k = 0.01; g = 9.81;
sys = {x''[t] == -k x'[t] Sqrt[x'[t]^2 + y'[t]^2],
       y''[t] == -k y'[t] Sqrt[x'[t]^2 + y'[t]^2] - g,
       x[0] == 0, y[0] == h, x'[0] == v0Cos[α], y'[0] == v0Sin[α]};
s = NDSolveValue[Join[sys,
       {WhenEvent[y[t] == 0, "StopIntegration"]}], {x, y}, {t, 0, ∞}]
tmax = s[[1, 1, 1, 2]]
ParametricPlot[Evaluate[{s[[1]][t], s[[2]][t]}], {t, 0, tmax}]   (рис. слева)
Animate[
ParametricPlot[Evaluate[{s[[1]][t], s[[2]][t]}], {t, 0, T},
            PlotRange → {{0, 0.12}, {0, 0.03}}], {T, 0, tmax}]
```



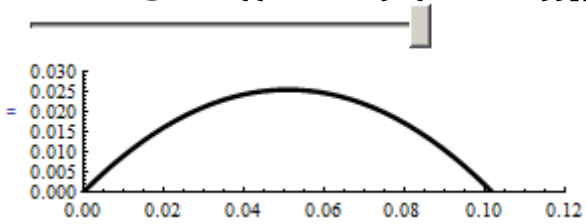


Чтобы для определения длительности полета нам не приходится подбирать значение `tmax` «на глазок», мы использовали функцию `WhenEvent` для определения момента наступления события «высота полета равна нулю» и задали реакцию на это событие `StopIntegration` (остановка вычислений). Момент времени, в который произошло событие остановки вычислений,

является параметром решения «InterpolationFunction» и доступ к нему мы получили с помощью индексации.

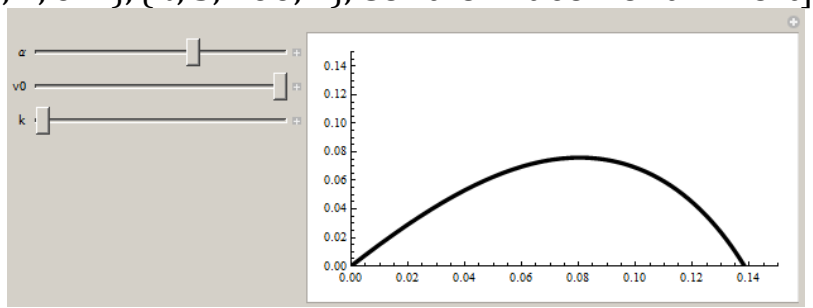
Вместо анимации можно использовать элемент «слайдер»

```
DynamicModule[{T =. 01},
  {Slider[Dynamic[T], {T, 0. 2}],
        Dynamic[
            ParametricPlot[Evaluate[{s[[1]][t], s[[2]][t]}], {t, 0, T},
                PlotRange → {{0, 0. 12}, {0, 0. 03}}]]}//TableForm]
```



Для наблюдением за поведением траектории тела при различных углах «бросания» α , начальных скоростях v_0 и коэффициентах сопротивления среды k удобно использовать манипулятор.

```
h = 0; g = 9. 81;
Manipulate[
  sys = {x''[t] == -kx'[t]√(x'[t]^2 + y'[t]^2),
        y''[t] == -ky'[t]√(x'[t]^2 + y'[t]^2) - g, x[0] == 0, y[0] == h};
   s = NDSolveValue[Join[sys, {x'[0] == v0Cos[α], y'[0] == v0Sin[α]},
          {WhenEvent[y[t] == 0, "StopIntegration"]}], {x, y}, {t, 0, ∞}];
  tmax = s[[1, 1, 1, 2]];
  ParametricPlot[Evaluate[{s[[1]][t], s[[2]][t]}], {t, 0, tmax},
       PlotRange → {{0, 0. 15}, {0, 0. 15}}, AspectRatio → 0. 5],
  {{α, π/4}, 0. 1, π/2 - 0. 06, 0. 1},
  {{v0, 1}, 0. 1, 2, 0. 1}, {k, 5, 100, 1}, ControlPlacement → Left]
```



4.5.10 Движение тел под действием тяготения.

Пример 10.1 Исследуем задачу о движении планеты вокруг Солнца под действием тяготения.

Закон всемирного тяготения Ньютона в векторной форме имеет вид $m\mathbf{a}=F\mathbf{e}$, где $\boldsymbol{a}$ – вектор ускорения, $\mathbf{e}$ – единичный вектор, направленный от планеты к солнцу, $F=\gamma\frac{mM}{R^2}$ – величина силы притяжения, R – расстояние между притягивающимися телами, m – масса планеты, M – масса солнца, γ – гравитационная постоянная. Свяжем начало координат с массивным телом (солнцем). В любой момент времени функции ($x(t)$,$y(t)$) представляют радиус-вектор планеты. Сила, с которой солнце притягивает планету, будет направлена вдоль единичного вектора

$$\mathbf{e}=\left(\frac{-x(t)}{\sqrt{x^2(t)+y^2(t)}},\frac{-y(t)}{\sqrt{x^2(t)+y^2(t)}}\right)$$

имеющего противоположное направление радиус-вектору планеты. Тогда

$$m\begin{Bmatrix}x''(t)\\y''(t)\end{Bmatrix}=-\gamma\frac{mM}{(x^2(t)+y^2(t))^{3/2}}\begin{Bmatrix}x(t)\\y(t)\end{Bmatrix},$$

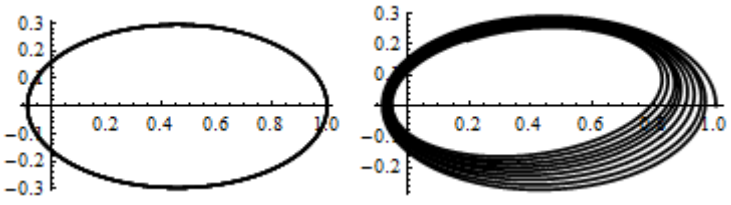
Введем обозначение $\gamma M=k$. Получим следующую систему обыкновенных дифференциальных уравнений относительно неизвестных функций $x(t)$, $y(t)$ – координат движущейся планеты:

$$\begin{cases}x''(t)=-\dfrac{k\,x(t)}{(x^2(t)+y^2(t))^{3/2}}\\[2ex] y''(t)=-\dfrac{k\,y(t)}{(x^2(t)+y^2(t))^{3/2}}\end{cases}$$

Для модельной задачи выберем k=1 и зададим начальные условия $x(0)=1,\ x'(0)=0,\ y(0)=0,\ y'(0)=1$. Решим задачу при погрешностях вычислений, заданных для функции `NDSolve` по – умолчанию, и при пониженных погрешностях.

```
sys = {x''[t] == -x[t]/(x[t]^2 + y[t]^2)^(3/2), y''[t] == -y[t]/(x[t]^2 + y[t]^2)^(3/2),
       x[0] == 1, x'[0] == 0, y[0] == 0, y'[0] == 0.4}
s = NDSolveValue[sys, {x, y}, {t, 0, 20}]
sb = NDSolveValue[sys, {x, y}, {t, 0, 20},
                  AccuracyGoal → 3, PrecisionGoal → 6]
ParametricPlot[{s[[1]][t], s[[2]][t]}, {t, 0, 20}, AspectRatio → Automatic]
ParametricPlot[{sb[[1]][t], sb[[2]][t]}, {t, 0, 20}, AspectRatio → Automatic]
```

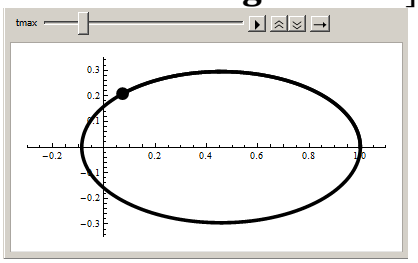
На следующем рисунке показана получаемая траектория движения планеты (кривая с параметрическим уравнением $x(t), y(t)$)



Левая траектория получена при стандартной точности вычислений, правая – при пониженной. В решении `sb` отчетливо видно как накопление погрешности вычислений влияет на результат.

Можно построить анимацию движения

```
s = NDSolveValue[sys, {x, y}, {t, 0, 20}]
Animate[
    p1 = ParametricPlot[Evaluate[{s[[1]][t], s[[2]][t]}], {t, 0, tmax},
          PlotRange → {{-0.3, 1.1}, {-0.35, 0.35}}, PlotPoints → 1000];
    p2 = Graphics[Disk[{s[[1]][tmax], s[[2]][tmax]}, 0.025]];
    Show[{p1, p2}],
    {tmax, 0, 20, 0.1}, AnimationRunning → False]
```



Пример 10.2 Смоделируем задачу о запуске искусственного спутника на орбиту Луны. В этом случае в уравнениях движения спутника под действием тяготения будет присутствовать внешняя сила $\mathbf{f}(t)$, действующая в течение ограниченного времени. Ее величину и длительность действия надо подобрать так, чтобы спутник вышел на орбиту Луны.

Отметим, что спутник должен подняться на достаточную высоту, чтобы его орбита не захватывала поверхность планеты. Поэтому внешняя сила должна вначале быть вертикальной (или иметь большую вертикальную составляющую), а затем быть горизонтальной, чтобы придать спутнику необходимую орбитальную скорость. Если вертикальная составляющая ускорения будет не достаточной, то траектория спутника (эллипс) будет задевать поверхность планеты.

В качестве центрального тела будем рассматривать Землю, с которой свяжем начало координат. Луна двигается под действием притяжения Земли и ее координаты обозначим $x_m(t), y_m(t)$. Координаты спутника обозначим $x_s(t), y_s(t)$, начальные значения которых должно быть на поверхности Земли. Притяжением Луны к спутнику можно пренебречь. Уравнения движения спутника будут иметь вид

$$\begin{cases} x_s''(t) = -\dfrac{\gamma M\, x_s(t)}{(x_s{}^2(t) + y_s{}^2(t))^{3/2}} + a_x(t) + \dfrac{\gamma m\left(x_s(t) - x_m(t)\right)}{\left(\left(x_s(t) - x_m(t)\right)^2 + \left(y_s(t) - y_m(t)\right)^2\right)^{3/2}} \\ y''(t) = -\dfrac{\gamma M\, y_s(t)}{(x_s{}^2(t) + y_s{}^2(t))^{3/2}} + a_y(t) + \dfrac{\gamma m\left(y_s(t) - y_m(t)\right)}{\left(\left(x_s(t) - x_m(t)\right)^2 + \left(y_s(t) - y_m(t)\right)^2\right)^{3/2}} \end{cases}$$

Здесь M – масса Земли, m – масса Луны, $\left(a_x(t), a_y(t)\right)$ - вектор ускорения, вызываемый внешней силой (работающим двигателем ракеты). Заметим, что

мы не учитываем факт изменения массы спутника, возникающего при сгорании топлива ракеты.

Луна также двигается под действием притяжения Земли и уравнения ее движения (такие же как в предыдущем примере) имеют вид

$$\begin{cases} x_m''(t) = -\dfrac{\gamma M\, x_m(t)}{(x_m{}^2(t) + y_m{}^2(t))^{3/2}} \\ y_m''(t) = -\dfrac{\gamma M\, y_m(t)}{(x_m{}^2(t) + y_m{}^2(t))^{3/2}} \end{cases}$$

В результате мы получаем систему четырех ОДУ, к которым надо добавить начальные условия. Здесь мы рассмотрим модельную задачу с вымышленными значениями масс и размеров.

Начальные условия спутника определяют, что в нулевой момент времени ракета-носитель спутника находится на поверхности планеты (радиуса 1) и имеет небольшую нормальную к поверхности скорость (вдоль оси X)

$$x_s(0) = 1, \quad y_s(0) = 0,\ x_s'(0) = 0.2,\ y_s'(0) = 0\,.$$

Начальные условия для Луны определяют ее положение и скорость

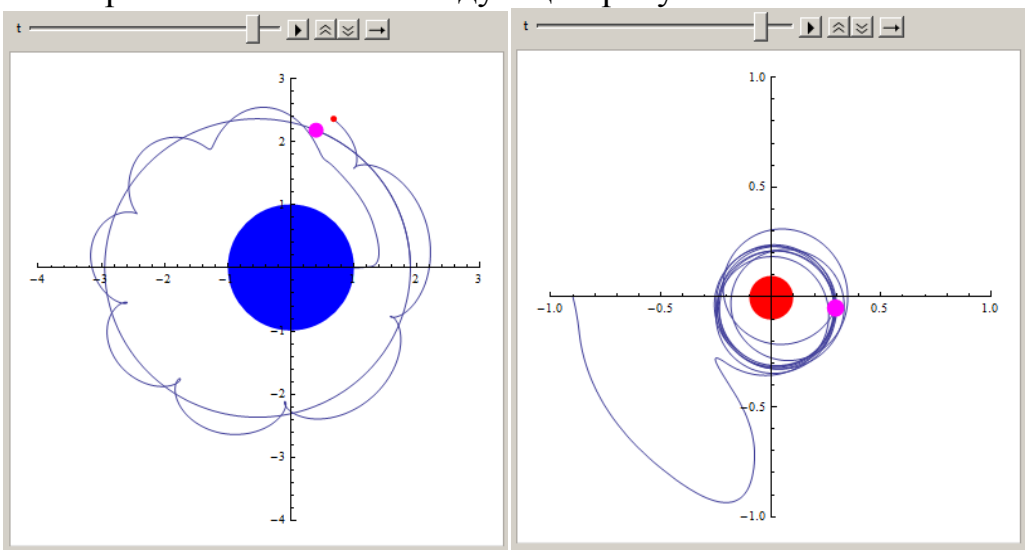
$$x_m(0) = 1.9, \quad y_m(0) = 0,\ x_m'(0) = 0,\ y_m'(0) = 0.8$$

Решим задачу и построим анимацию движения Луны и спутника, а также анимацию видимого движения спутника с поверхности Луны.

```
Remove[xs, ys, xm, ym, x, y, x1, y1]
M = 10; m = 1; γ = 0.1; t0 = 30;
ax[t_] = Piecewise[{{0, t ≤ 0.2}, {1.4, t ≤ 1}, {-0.5, t ≤ 2}, {-0.9, t
          ≤ 2.5}, {1, t ≤ 3.1}, {0, t > 3.1}}];
ay[t_] = Piecewise[{{0, t ≤ 0.2}, {0, t ≤ 1}, {1.2, t ≤ 2}, {-0.2, t ≤ 2.5}, {-2, t
          ≤ 3.1}, {0, t > 3.1}}];
eq1 = xs''[t] == -γ M xs[t] / (xs[t]^2 + ys[t]^2)^(3/2) + ax[t] +
                 γ m (xm[t] - xs[t]) / ((xm[t] - xs[t])^2 + (ym[t] - ys[t])^2)^(3/2);
eq2 = ys''[t] == -γ M ys[t] / (xs[t]^2 + ys[t]^2)^(3/2) + ay[t] +
                 γ m (ym[t] - ys[t]) / ((xm[t] - xs[t])^2 + (ym[t] - ys[t])^2)^(3/2);
eq3 = xm''[t] == -γ M xm[t] / (xm[t]^2 + ym[t]^2)^(3/2);
eq4 = ym''[t] == -γ M ym[t] / (xm[t]^2 + ym[t]^2)^(3/2);
bc = {xs[0] == 1, ys[0] == 0, xs'[0] == 0.2, ys'[0] == 0,
      xm[0] == 1.9, ym[0] == 0, xm'[0] == 0, ym'[0] == 0.8};
sys = Join[{eq1, eq2, eq3, eq4}, bc];
s = NDSolve[sys, {xs, ys, xm, ym}, {t, 0, t0}]
```

```
x = xs/.s[[1,1]]; y = ys/.s[[1,2]];
x1 = xm/.s[[1,3]]; y1 = ym/.s[[1,4]];
Animate[
  pp = ParametricPlot[{x[τ], y[τ]}, {τ, 0, t}, PlotRange → {{-4, 3}, {-4, 3}}];
  ss = ParametricPlot[{x1[τ], y1[τ]}, {τ, 0, t}, PlotRange → {{-4, 3}, {-4, 3}}];
  ssd = Graphics[{RGBColor[1, 0, 0], Disk[{x[t], y[t]}, 0.05]}];
  gr = Graphics[{RGBColor[1, 0, 1], Disk[{x1[t], y1[t]}, 0.12]}];
  earth = Graphics[{RGBColor[0, 0, 1], Disk[{0, 0}, 1]}];
  Show[{pp, ssd, ss, gr, earth}, Axes → True],
  {t, 0.1, 30, 0.05}, AnimationRunning → False]
```

Результат построения показан на следующем рисунке слева.



Следующий код строит траекторию движения спутника, наблюдаемую с Луны.

```
Animate[
  pp = ParametricPlot[{x[τ] - x1[τ], y[τ] - y1[τ]}, {τ, 0, t},
                      PlotRange → {{-1, 1}, {-1, 1}}];
  ssd = Graphics[{RGBColor[1, 0, 0], Disk[{0, 0}, 0.1]}];
  gr = Graphics[{RGBColor[1, 0, 1], Disk[{x[t] - x1[t], y[t] - y1[t]}, 0.04]}];
  Show[{pp, ssd, gr}, Axes → True],
  {t, 0.1, 30, 0.05}, AnimationRunning → False]
```

Траектория спутника, наблюдаемая с Луны, показана на предыдущем рисунке справа.

4.5.11 Математический маятник.

Исследуем поведения математического маятника. Пусть масса груза равна единице, а стержень, на котором подвешена масса, невесом. Тогда дифференциальное уравнение движения груза имеет вид

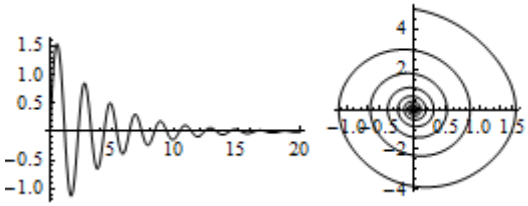
$$\varphi'' + k\,\varphi' + \omega^2 \sin\varphi = 0$$

где $\varphi(t)$ угол отклонения маятника от положения равновесия (нижнее положение), параметр k характеризует величину трения, $\omega^2 = g/l$ (g ускорение свободного падения, l – длина маятника). Для определения конкретного

движения к уравнению движения надо добавить начальные условия $\varphi(0)=\varphi_0$, $\varphi'(0)=\varphi'_0$. Выберем следующие значения параметров k=0.5, $\omega^2=10$ и начальные значения $\varphi_0=0, \varphi'_0=5$.

Решаем задачу, строим график решения (следующий рисунок слева) и фазовую траекторию (рисунок справа)

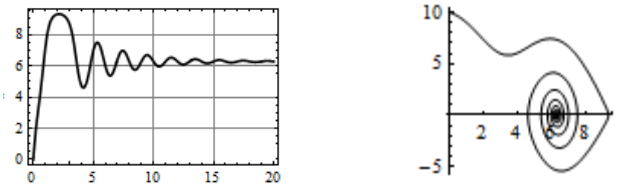
```
k = 0.5; w = 10;
s = NDSolveValue[{x''[t] + kx'[t] + wSin[x[t]] == 0, x[0] == 0,
                                                x'[0] == 5}, x, {t, 0, 20}]
Plot[s[t], {t, 0, 20}]
ParametricPlot[Evaluate[{s[t], s'[t]}], {t, 0, 20}, AspectRatio → 1]
```



Как видно из левого графика, максимальный угол отклонения маятника не превышают $\pi/2$ и колебания маятника затухают.

Увеличим начальную скорость до 10. Решаем задачу, строим график решения (следующий рисунок слева) и фазовую траекторию (рисунок справа)

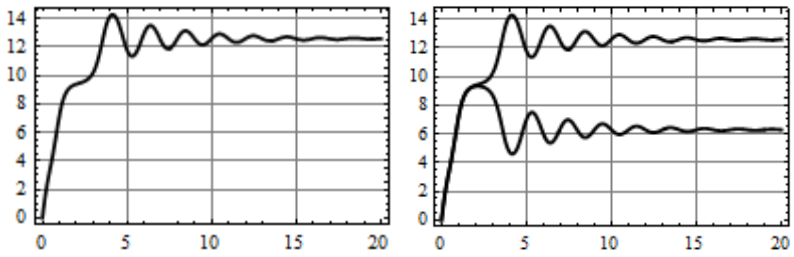
```
s = NDSolveValue[{x''[t] + kx'[t] + wSin[x[t]] == 0, x[0] == 0,
                                                x'[0] == 10}, x, {t, 0, 20}]
Plot[s[t], {t, 0, 20}]
p1=ParametricPlot[Evaluate[{s[t], s'[t]}], {t, 0, 20}, AspectRatio → 1]
```



Максимальное значение угла составляет примерно 10 радиан. Маятник сделал один полный оборот вокруг точки закрепления (угол отклонения увеличился на $2\cdot\pi$), а затем колебания затухают в окрестности значения $2\cdot\pi$ (для маятника угол поворота $2\cdot\pi$ представляет то же, что и 0 радиан, т.е. положение равновесия).

Обратим внимание на следующий факт – точность вычислений в последнем примере имеет существенное значение. Снизим относительную и абсолютную погрешности опциями `AccuracyGoal→3, PrecisionGoal→3` и найдем решение той же задачи

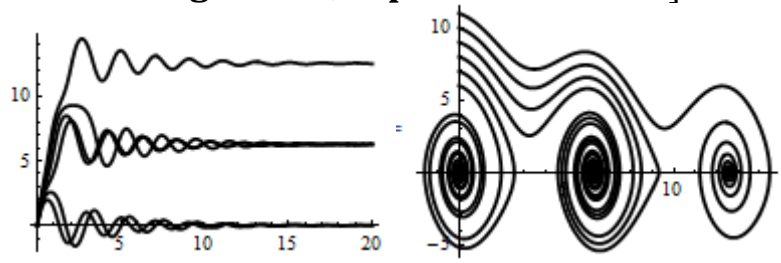
```
sbad = NDSolveValue[{x''[t] + kx'[t] + wSin[x[t]] == 0,
  x[0] == 0, x'[0] == 10}, x, {t, 0, 20}, AccuracyGoal → 3, PrecisionGoal →
3]
p2 = Plot[sbad[t], {t, 0, 20}, GridLines → Automatic,
                                    Frame → True, PlotRange → All]
Show[p1, p2]
```



На правом рисунке показаны оба графика решений: второй более высокий, полученный при пониженной точности вычислений, и первый – полученный при стандартной точности. Отличие весьма значительное. Верхняя кривая представляет маятник, сделавший два полных оборота вокруг точки подвеса, а нижняя – маятник, сделавший только один полный оборот. Изменение точности привело к «качественному» изменению решения – два оборота маятника вместо одного!

Построим несколько графиков угла отклонения (след. рис. слева) и фазовых траекторий (след. рис. справа), задавая различную начальную скорость.

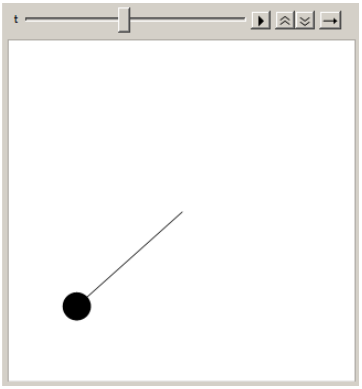
```
k = 0.5; w = 10;
s = ParametricNDSolveValue[{x''[t] + kx'[t] + wSin[x[t]] == 0,
                            x[0] == 0, x'[0] == v0}, x, {t, 0, 20}, {v0}]
Plot[Table[s[v0][t], {v0, 6, 11, 1}], {t, 0, 20},
                            PlotStyle → {Thickness[0.01], {Black}}]
ParametricPlot[Table[{s[v0][t], s[v0]'[t]}, {v0, 6, 11, 1}], {t, 0, 20},
                            PlotRange → All, AspectRatio → 0.7]
```



Как видим, начальная скорость при *v=6, 7* недостаточна, чтобы маятник прошел верхнюю точку и сделал хотя бы один полный оборот. При начальной скорости *v= 8, 9, 10* маятник совершает один полный оборот, а затем его колебания затухаю. При *v=11* маятник смог выполнить два полных оборота и только после этого его колебания стали затухать вокруг положения равновесия.

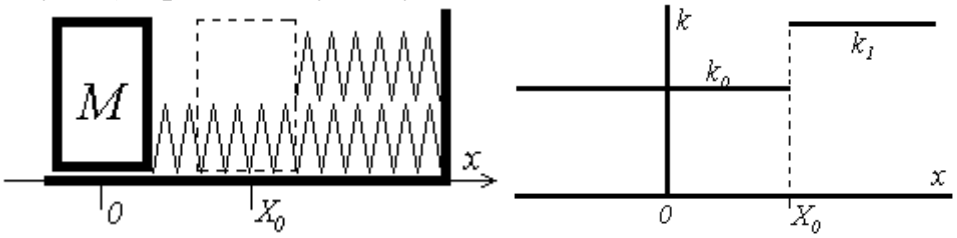
Для наглядности, построим еще анимацию движения модели маятника

```
φ = NDSolveValue[{x''[t] + kx'[t] + wSin[x[t]] == 0,
                            x[0] == 0, x'[0] == 10}, x, {t, 0, 20}]
g1[t_] = Line[{{0, 0}, {-Sin[φ[t]], -Cos[φ[t]]}}];
d1[t_] = Disk[{-Sin[φ[t]], -Cos[φ[t]]}, 0.1];
Animate[Graphics[{g1[t], d1[t]}, PlotRange → {{-1.1, 1.1}, {-1.1, 1.1}}],
                            {t, 0, 4π}, AnimationRunning → False]
```



4.5.12 Колебание массивного тела под действием пружин.

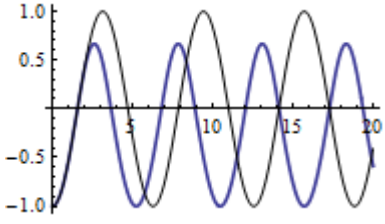
Пример 12.1 Решим задачу о колебании массивного тела массы M на невесомых пружинах, одна из которых короче другой и короткая не привязана к телу (см. рисунок), трение отсутствует.



Для тела M выполняется уравнение движения $\ddot{x}(t)+k^2x(t)=0$. Здесь коэффициент k отвечает за жесткость пружины. При движении вправо и достижении положения X_0 тело M присоединяется ко второй пружине и жесткость системы меняется. Когда тело движется влево, и проходит положение X_0, оно отрывается от второй пружины и жесткость уменьшается. Т.о. жесткость зависит от смещения груза M относительно положения равновесия, т.е. $k=k(x)$. При этом функция $k(x)$ является кусочно постоянной. График функции $k(x)$ приведен на предыдущем рисунке справа.

Пусть $k_0=1$, $k_1=1.5$ и начальные значения $x(0)=-1$, $x'(0)=v_0=1$. Для сравнения решим задачу при постоянном k=1 и при кусочно – постоянной жесткости k. Тогда

```
uc = NDSolve[{x''[t] + x[t] == 0, x[0] == -1, x'[0] == 0}, x, {t, 0, 20}]
up = NDSolve[{x''[t] + (1 + 0.5UnitStep[x[t]])^2 x[t] == 0,
              x[0] == -1, x'[0] == 0}, x, {t, 0, 20}]
Plot[{x[t]/. uc[[1, 1]], x[t]/. up[[1, 1]]}, {t, 0, 20}]
ParametricPlot[{Evaluate[{x[t]/. up[[1, 1]], x'[t]/. up[[1, 1]]}],
                Evaluate[{x[t]/. uc[[1, 1]], x'[t]/. uc[[1, 1]]}]}, {t, 0, 20}]
```



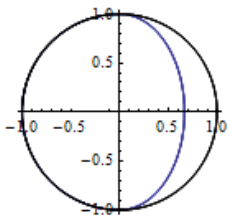


График решения показан на следующем рисунке синей (более низкой) кривой. Черная (более высокая кривая) представляет решение для случая неизменного

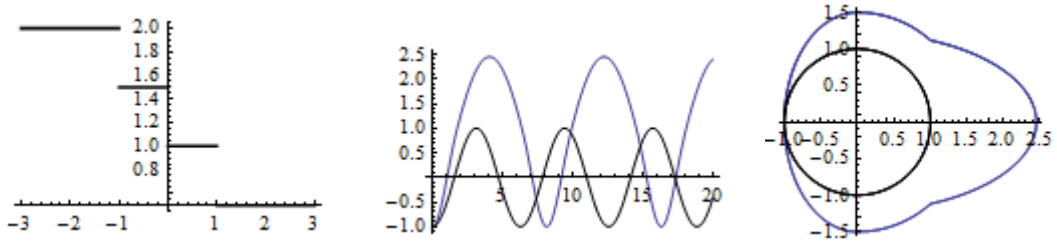
k=1. На правом рисунке показана фазовая траектория синей кривой, черная (окружность) представляет фазовую траекторию для k=1.

Здесь для функции $k(x)$ мы выбрали представление 1+0.5 UnitStep[x[t]]. Допустимо использование функции Хэвисайда 1+0.5 HeavisideTheta[x[t]]. Можно также создать свою кусочную функцию f[x_]=Piecewise[{{1,x<0}},1.5] и использовать ее при решении уравнения, например, так

$$\mathbf{up} = \mathbf{NDSolve}[\{x''[t] + f[x[t]]^2 x[t] == 0, \\ x[0] == -1, x'[0] == 0\}, x, \{t, 0, 20\}]$$

В последнем случае кусков функции f можно задать много.

$$f[\mathrm{x\_}] = \mathbf{Piecewise}[\{\{2, x < -1\}, \{1.5, x < 0\}, \{1, x < 1\}\}, 0.5];$$
$$\mathbf{Plot}[f[x], \{x, -3, 3\}]$$
$$\mathbf{uc} = \mathbf{NDSolve}[\{x''[t] + x[t] == 0, x[0] == -1, x'[0] == 0\}, x, \{t, 0, 20\}]$$
$$\mathbf{up} = \mathbf{NDSolve}[\{x''[t] + f[x[t]]^2 x[t] == 0, \\ x[0] == -1, x'[0] == 0\}, x, \{t, 0, 20\}]$$
$$\mathbf{Plot}[\{x[t]/.\mathbf{up}[[1,1]], x[t]/.\mathbf{uc}[[1,1]]\}, \{t, 0, 20\}]$$
$$\mathbf{ParametricPlot}[\{\mathbf{Evaluate}[\{x[t]/.\mathbf{up}[[1,1]], x'[t]/.\mathbf{up}[[1,1]]\}], \\ \mathbf{Evaluate}[\{x[t]/.\mathbf{uc}[[1,1]], x'[t]/.\mathbf{uc}[[1,1]]\}]\}, \{t, 0, 20\}, \\ \mathbf{AspectRatio} \to \mathbf{Automatic}, \mathbf{PlotPoints} \to 1000]$$



На предыдущем рисунке слева показана, созданная нами кусочная функция, в середине – график решения (синяя кривая), справа – фазовая траектория (синяя кривая).

Пример 12.2. Рассмотрим систему, показанную на следующем рисунке. Груз массы m находится посередине, обе пружины касаются его, но не соединяются с ним. Когда груз находится посередине, растяжение пружин равно нулю. Жесткости пружин разные и их массой можно пренебречь. При смещении груза из положения равновесия возникают колебания.

Уравнение движения груза при $x < 0$ имеет вид

$$\ddot{x} + \frac{c_1}{m} x = 0, \quad x < 0$$

Уравнение движения груза при $x > 0$ имеет вид

$$\ddot{x} + \frac{c_2}{m} x = 0, \;\; x > 0$$

Таким образом, единое уравнение можно записать в виде

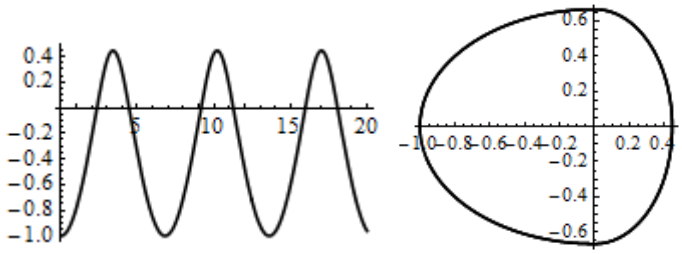
$$\ddot{x} + k^2 x = 0, \qquad (1)$$

где

$$k^2(x) = \begin{cases} c_1 / m, \, x < 0 \\ c_2 / m, \, x > 0 \end{cases} = \begin{cases} k_1^2, \, x < 0 \\ k_2^2, \, x > 0 \end{cases} = k_1^2 + \left(k_2^2 - k_1^2\right) H(x)$$

где $H(x)$ – функция Хэвисайда. Обратите внимание, что коэффициент $k(x)$ зависит от решения $x(t)$. Функция `DSolve` не может решить эту задачу, поэтому используем `NDSolve`.

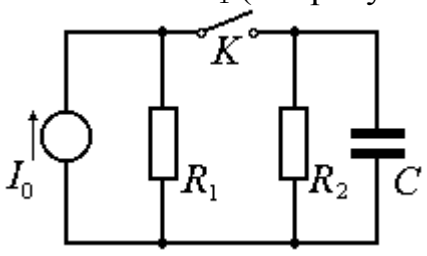
```
k1 = 2/3; k2 = 3/2; k[x_] = Piecewise[{{k1^2, x < 0}, {k2^2, x >= 0}}];
s = NDSolve[{x''[t] + k[x[t]] x[t] == 0, x[0] == -1, x'[0] == 0}, x, {t, 0, 20}]
Plot[s[[1, 1, 2]][t], {t, 0, 20}]
ParametricPlot[Evaluate[{s[[1, 1, 2]][t], D[s[[1, 1, 2]][t], t]}], {t, 0, 20}]
```



Слева показан график зависимости решения от времени, справа – фазовая траектория.

4.5.13 Генератор пилообразных напряжений.

Пример 13.1. Простой генератор напряжений пилообразной формы состоит из параллельно соединенных емкости C и сопротивления R_2, которые периодически подключаются к параллельно соединенным генератором постоянного тока I_0 и сопротивлением R_1 (см. рисунок).



Коммутирующее устройство, периодически осуществляющее включение и отключение, ради простоты на схеме представлено в виде рубильника K, который замкнут в течение первой части цикла длительностью t_1 секунд и разомкнут в течение второй части периода длительностью T-t_1 секунд. Задача состоит в определении закона нарастания напряжения между обкладками конденсатора C (в начальный момент конденсатор разряжен).

Здесь мы рассмотрим не периодический процесс, а только один шаг, состоящий во включении и отключении рубильника. Обозначим через $u_1(t)$ напряжение конденсатора при замкнутом рубильнике, т.е. в интервале времени $0=t_0<t<t_1$, и $u_2(t)$ напряжение на конденсаторе при разомкнутом рубильнике для моментов времени $t>t_1$.

Дифференциальные уравнения для разных интервалов времени имеют вид

$$C\frac{d\,u_1}{d\,t}+\frac{1}{R_0}u_1=I_0 \text{ для } 0\le t\le t_1;$$

$$C\frac{d\,u_2}{d\,t}+\frac{1}{R_2}u_2=0 \text{ для } t\ge t_1,$$

где $\frac{1}{R_0}=\frac{1}{R_1}+\frac{1}{R_2}$. Так как напряжение на конденсаторе может меняться только непрерывно, то значение u_1 в момент t_1 равно начальному значению u_2 в этот же момент времени $u_1(t_1)=u_2(t_1)$. Оба эти уравнения мы можем представить в виде одного ДУ $\frac{d\,u}{d\,t}+a(t)\cdot u=f(t)$, где $a(t)$ и $f(t)$ кусочно-постоянные функции.

На участке $0\le t<t_1$ функция $a(t)$ равна $a_1=\frac{1}{C\,R_0}$, и для $t\ge t_1$ равна $a_2=\frac{1}{C\,R_2}$.

Функция $f(t)$ на участке $0\le t<t_1$ равна $f_1=\frac{I_0}{C}$, а для $t\ge t_1$ равна $f_2=0$.

Соответственно функция $u(t)$ на этих временных участках равна $u_1(t)$ и $u_2(t)$. Т.о. мы приходим к задаче

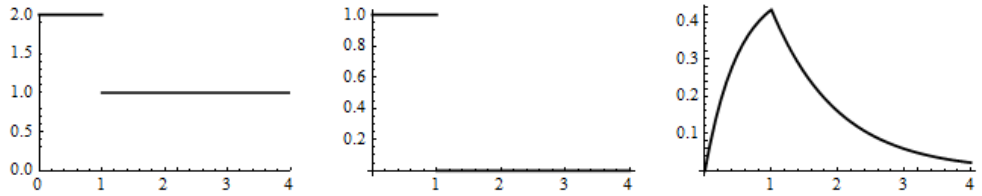
$$\frac{d\,u}{d\,t}+a(t)\cdot u=f(t),\ u(0)=0,$$

где

$$a(t)=\begin{cases}a_1\,,\ t<t_1\\ a_2\,,\ t>t_1\end{cases} \qquad \text{и} \qquad f(t)=\begin{cases}f_1\,,\ t<t_1\\ 0\,,\ t>t_1\end{cases}$$

Теперь можно написать код решения задачи (положим C=1).

```
R1 = 1; R2 = 1; I0 = 1; f1 = I0; t1 = 1;
R0 = R1 R2/(R1 + R2); a1 = 1/R0; a2 = 1/R2;
a[t_] = Piecewise[{{a1, t < t1}}, a2];
f[t_] = Piecewise[{{f1, t < t1}}, 0];
sU = DSolve[{u'[t] + a[t]u[t] == f[t], u[0] == 0}, u, t];
p1 = Plot[a[t], {t, 0, 4}, PlotRange → {{0, 4}, {0, 2}}];
p2 = Plot[f[t], {t, 0, 4}];
p3 = Plot[sU[[1, 1, 2, 2]], {t, 0, 4}];
GraphicsRow[{p1, p2, p3}]
```



Слева показан график функции $a(t)$, в середине – функции $f(t)$, справа – график решения.

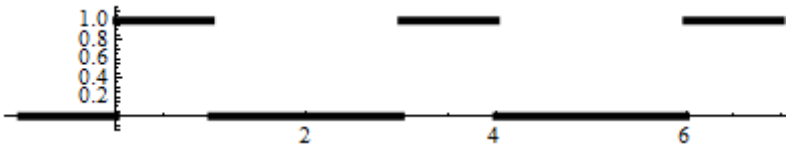
Пример 13.2. Пусть рубильник включается и выключается периодически. Это значит, что функции $a(t)$ и $f(t)$ должны совпадать с одноименными функциями из предыдущего примера на отрезке периода T и затем повторяться периодически. Для моделирования кусочно – постоянной периодической функции создадим одну специальную функцию, которую назовем *Periodic Step Function = Psf*

$$Psf(x,a,w)=\left[\frac{x}{w}\right]-\left[\frac{x-a}{w}\right]$$

Это периодическая с периодом w функция. Для $a<w$ на отрезке $0\le x<a$ функция равна 1, на остальном куске периода $a\le x<w$ функция равна 0. Случай $a\ge w$ мы не рассматриваем. На следующем рисунке приведен график функции *Psf*(*x*,*1*,*3*).

$$\mathbf{Psf[x\_,a\_,w\_]=Floor\left[\frac{x}{w}\right]-Floor\left[\frac{x-a}{w}\right];}$$

$$\mathbf{Plot[Psf[x,1,3],\{x,-1,7\},AspectRatio\rightarrow Automatic]}$$



Эта импульсивная функция принимает значения +1 и 0. Используя эту функцию, можно смоделировать периодические функции, совпадающие с $a(t)$ и $f(t)$ на отрезке периода T. Пусть T=3. Тогда создадим требуемые функции и решим соответствующее ОДУ

$$\mathbf{Remove[t,a,f,u];\ t1=1;\ T=3;}$$

$$\mathbf{a[t\_]=1+Psf[t,t1,T];}$$

$$\mathbf{f[t\_]=Psf[t,t1,T];}$$

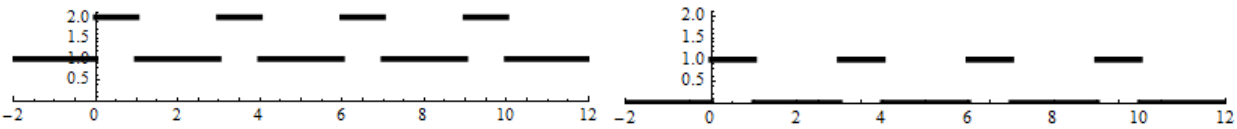
$$\mathbf{Plot[a[t],\{t,-2,12\},PlotRange\rightarrow\{\{-2,12\},\{0,2.1\}\}]}$$

$$\mathbf{Plot[f[t],\{t,-2,12\},PlotRange\rightarrow\{\{-2,12\},\{0,2.1\}\}]}$$

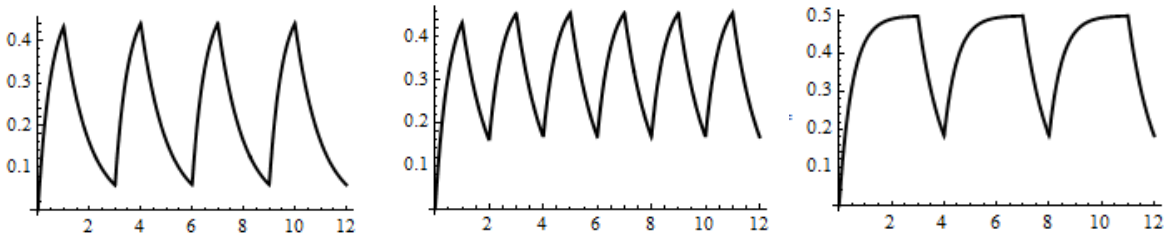
$$\mathbf{nds=NDSolve[\{u'[t]+a[t]u[t]==f[t],u[0]==0\},u,\{t,0,12\}]}$$

$$\mathbf{xs=u/.nds[[1]];}$$

$$\mathbf{Plot[xs[t],\{t,0,12\}]}$$



На предыдущем рисунке слева показана периодическая кусочно – постоянная функция $a(t)$, а справа – периодическая кусочно – постоянная функция $f(t)$. На следующем рисунке слева показан график решения.



В предыдущем решении время включения рубильника составляло треть времени одного периода ($t_1 = 1; T = 3$). Если положить $t_1 = 1; T = 2$, то график решения будет таким, как показано на предыдущем рисунке в середине. Для его построения в предыдущем коде вы должны параметру T присвоить другое значение, например, $T = 2$. В случае $t_1 = 3; T = 4$ график решения получится таким, как показано на рисунке справа.

4.5.14 Двухвидовая модель Вольтерра «хищник – жертва».

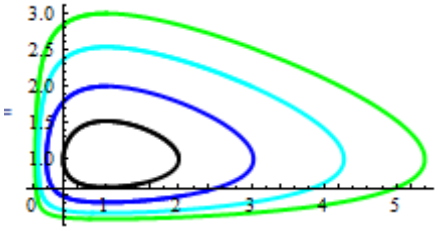
Рассмотрим двухвидовую модель «хищник – жертва», впервые построенную Вольтерра для объяснения колебаний рыбных уловов. Имеются два биологических вида, численностью в момент времени `t`, соответственно, `x(t)` и `y(t)`. Особи первого вида являются пищей для особей второго вида (хищников). Численности популяций в начальный момент времени известны. Требуется определить численность видов в произвольный момент времени. Математической моделью задачи является система дифференциальных уравнений Лотки – Вольтерра

$$\begin{cases} \dfrac{dx}{dt} = (a - b\,y)\,x \\ \dfrac{dy}{dt} = (-c + d\,x)\,y \end{cases}$$

где `a,b,c,d` – положительные константы. Проведем расчет численности популяций при $a = 3, b = 3, c = 1, d = 1$, для нескольких вариантов начальных условий: $x(0) = 2, y(0) = 1$; $x(0) = 1, y(0) = 2$; $x(0) = 1, y(0) = 3$; $x(0) = 3, y(0) = 2$. Организуем начальные условия в виде списка и построим для них фазовые траектории.

```
a = 3; b = 3; c = 1; d = 1;
sys: = {x'[t] == (a - by[t])x[t], y'[t] == (-c + dx[t])y[t]}
bc := {{x[0] == 2, y[0] == 1}, {x[0] == 1, y[0] == 2},
                {x[0] == 1, y[0] == 3}, {x[0] == 3, y[0] == 2}}
col: = {{Black}, {Blue}, {Green}, {Cyan}}
pp = Table[se = Join[sys, bc[[i]]];
          u = NDSolve[se, {x, y}, {t, 0, 7}];
          fx = u[[1, 1, 2]];
          fy = u[[1, 2, 2]];
          p = ParametricPlot[{fx[t], fy[t]}, {t, 0, 7},
                    PlotStyle → {Thickness[0.01], col[[i]]}];
```

```
        p, {i, 1, 4}];
Show[pp, PlotRange → All]
```

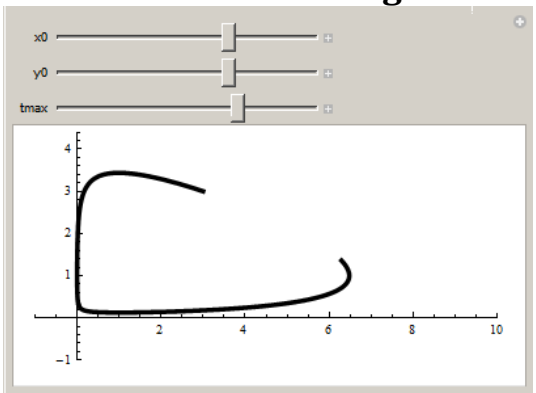


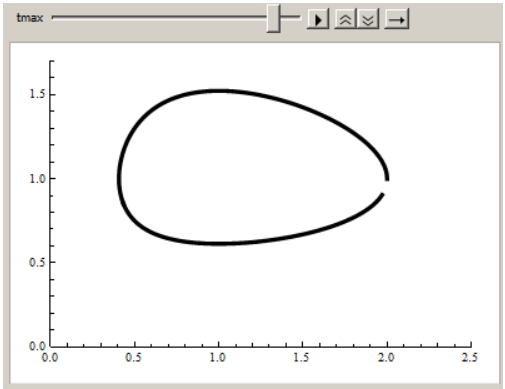
Из вида фазовых траекторий видно, что численность популяций меняется периодически. Для более детального анализа фазовых траекторий можно использовать «манипулятор» (следующий рисунок слева). В нем мы будем менять начальные значения x_0, y_0 искомых функций и интервал времени t_{max}.

```
a = 3; b = 3; c = 1; d = 1;
Module[{x, y, sys, se, u, fx, fy},
  sys: = {x'[t] == (3 − 3y[t])x[t], y'[t] == (−1 + x[t])y[t]};
  Manipulate[
    se = Join[sys, {x[0] == x0, y[0] == y0}];
    u = NDSolve[se, {x, y}, {t, 0, tmax}];
    ParametricPlot[Evaluate[{fx[t], fy[t]}], {t, 0, tmax},
                   PlotStyle → {Thickness[0.01], {Black}},
                  PlotRange → {{−1, 10}, {−1, 5}}],
    {x0, 1, 4, 0.5}, {y0, 1, 4, 0.5}, {tmax, 0.05, 7, 0.05},
    AutoAction → False, Initialization: >
    {fx: = u[[1, 1, 2]]; fy: = u[[1, 2, 2]]}]]
```

Можно построить анимацию (следующий рисунок справа)

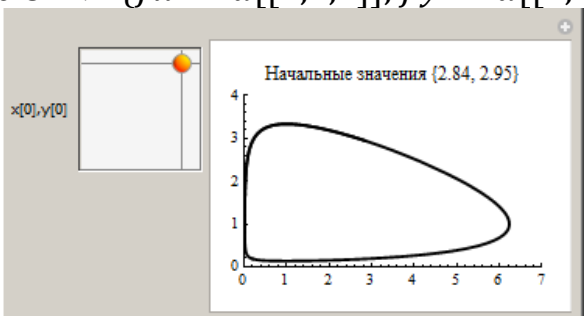
```
Animate[
    u = NDSolve[{x'[t] == (a − by[t])x[t], y'[t] == (−c + dx[t])y[t],
    x[0] == 2, y[0] == 1}, {x, y}, {t, 0, tmax}];
    fx = u[[1, 1, 2]];
    fy = u[[1, 2, 2]];
    ParametricPlot[Evaluate[{fx[t], fy[t]}], {t, 0, tmax},
          PlotStyle → {Thickness[0.01], {Black}},
          PlotRange → {{0, 2.5}, {0, 1.7}}], {tmax, 0.01, 4, 0.01},
         AnimationRunning → False]
```





Можно использовать «2D манипулятор» для одновременного изменения начальных значений популяций.

```
Module[{u, se, x, y, pt, sys, fx, fy},
  a = 3; b = 3; c = 1; d = 1;
  sys = {x'[t] == (a - by[t])x[t], y'[t] == (-c + dx[t])y[t]};
  Manipulate[
    se:= Join[sys, {x[0] == pt[[1]], y[0] == pt[[2]]}];
    u = NDSolve[se, {x, y}, {t, 0, 7}];
    ParametricPlot[{fx[t], fy[t]}, {t, 0, 7},
                PlotStyle → {Thickness[0.01], Black},
                PlotRange → {{0, 7}, {0, 4}}, PlotLabel →
                  StringJoin["Начальные значения", ToString[pt]]],
    {{pt, {2, 2},x[0],y[0]}, {1, 1}, {3, 3}},
    ControlPlacement → Left, AutoAction → False,
          Initialization:> {fx:= u[[1,1,2]]; fy:= u[[1,2,2]]; }]]
```



Весь код мы включили в функцию `Module` для того, чтобы основные переменные кода, стоящие в списке первого аргумента и являющиеся локальными для модуля, не взаимодействовали с одноименными переменными блоков кода других примеров. Имеет смысл коды всех примеров обрамлять этой функцией.

Замечания о выполнении примеров.

Если вы начинаете новую сессию работы с новым документом, то ввод и выполнение кода любого примера пройдет гладко так, как описано нами. Все примеры протестированы в системе Mathematica 9. Если вы работаете с версией *Mathemanica* 6, 7 или 8, то в них нет некоторых функций, которые мы описывали и использовали в примерах. Например, в прежних версиях системы нет функций `DifferentialRoot`, `DifferentialRootReduce`, `NDSolveValue`, `ParametricNDSolve`, `ParametricNDSolveValue` и некоторых связанных с ними функций. В примерах с функцией `NDSolveValue` вы можете выполнить замену на функцию `NDSolve`. Конечно придется выполнить и некоторые изменения последующего кода, поскольку результаты, возвращаемые этими функциями, используются по-разному. Коды, содержащие другие из указанных функций, вам придется существенно переработать. В версиях *Mathemanica* 4 и 5, кроме прочего, нет управляющих элементов `Manipulate` и других. Также в новых версиях

системы некоторые опции графических функций переработаны и имеют другие названия.

Более существенные замечания касаются случая, когда в одном документе вы будете выполнять множество из приведенных нами примеров. В этом случае будет возникать конфликт имен переменных, которые, в целях краткости, мы называем одинаково *x*, *y*, *t* и т.д. Если перед кодом примера вы будете добавлять команду `Remove[имя1, имя2, ...]` , где `имя1, имя2,` это имена переменных, используемых в коде, то в большинстве случаев этого достаточно для корректного выполнения текущего примера. Однако, в таком случае выполнение текущего примера может приводить к порче результатов некоторых примеров, выполненных вами ранее. Чтобы защитить код примера от влияния других примеров, его (код) следует включить в тело функции `Module` или `DynamicModule`. Первым аргументом этих функций должен быть список имен защищаемых (локализуемых в блоке) переменных, а вторым – код примера, команды которого нужно (обязательно) разделять точкой с запятой. При этом функцию `DynamicModule` нужно использовать тогда, когда в коде примера имеются динамические переменные. С целью экономии в большинстве примеров этого пособия мы эти функции не использовали.