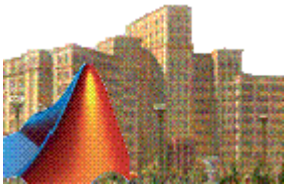


Доля П.Г.
Харьковский Национальный Университет
механико – математический факультет
кафедра геометрии им. А.В. Погорелова

Использование MATLAB. Решение дифференциальных уравнений в частных производных. PDE Toolbox.

В предыдущих частях пособия были подробно рассмотрены основные элементы, необходимые для уверенного использования системы MatLab. После его прочтения можно решать практически любые инженерные и математические задачи. Однако в MalLab есть еще большое количество пакетов расширения, которые многократно увеличивают эффективность использования системы. Одним из таких пакетов является PDE TOOLBOX, предназначенный для решения дифференциальных уравнений в частных производных (ДУЧП) и их систем. В данной части мы приводим основные сведения, которые помогут вам использовать возможности этого пакета, а также рассматриваем другие способы решения таких уравнений и их систем.

Оглавление

Часть 4. Решение дифференциальных уравнений в частных производных. PDE Toolbox.

4.1 ВВЕДЕНИЕ В PDE TOOLBOX

Многие прикладные задачи сводятся к решению дифференциальных уравнений в частных производных (ДУЧП) и их систем. Одним из наиболее распространенных приближенных методов их решения является метод конечных элементов (МКЭ). Данная глава посвящена описанию возможностей пакета расширения Partial Differential Equations Toolbox (PDE Toolbox), предназначенного для решения краевых задач для дифференциальных уравнений в частных производных в двумерных областях методом конечных элементов.

Пакет состоит из набора функций, автоматизирующих реализацию МКЭ для решения различного типа ДУЧП 2 – го порядка и их систем: эллиптических, параболических и гиперболических. Кроме того, в состав

пакета входит приложение `pdetool` с графическим интерфейсом пользователя, использование которого не требует глубокого понимания метода конечных элементов и упрощает доступ к набору функций пакета.

4.1.1 Графический интерфейс PDE Toolbox

Решение краевых задач для отдельных уравнений.

Рассмотрим пример решения задачи

$$\Delta u = 16 \cdot \left(x^2 + y^2\right), \quad u\big|_{\partial\Omega} = 1,$$

где область Ω представляет собой круг единичного радиуса с центром в начале координат. Выбор этой задачи объясняется тем, что ее точное решение известно $u = \left(x^2 + y^2\right)^2$, и мы сможем сравнить его с решением, полученным с помощью PDE TOOLBOX.

Наберите в командном окне MalLab команду

pdetool

Откроется окно программы PDE Toolbox

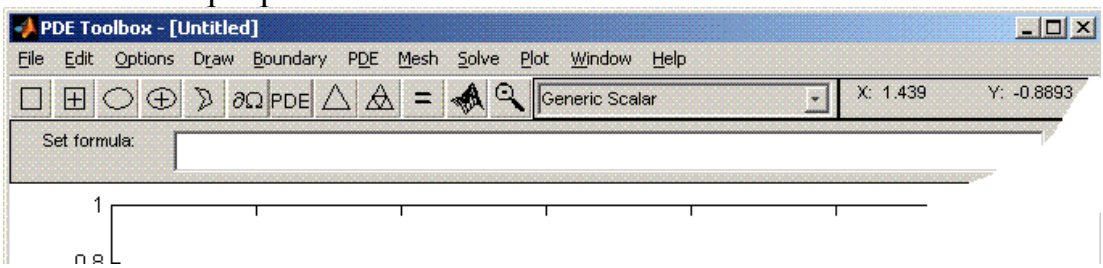


Рис. 1

Используя инструменты этого окна, мы можем ввести исходные данные, найти приближенное решение задачи и построить его график.

Ввод условий задачи осуществляется в четыре этапа:

1. **Задание двумерной области Ω**. Нажмите кнопку, на которой нарисован эллипс с плюсом внутри  , и при помощи мыши нарисуйте эллипс. Затем дважды щелкните мышью по появившейся области. Появится диалоговое окно с параметрами эллипса.

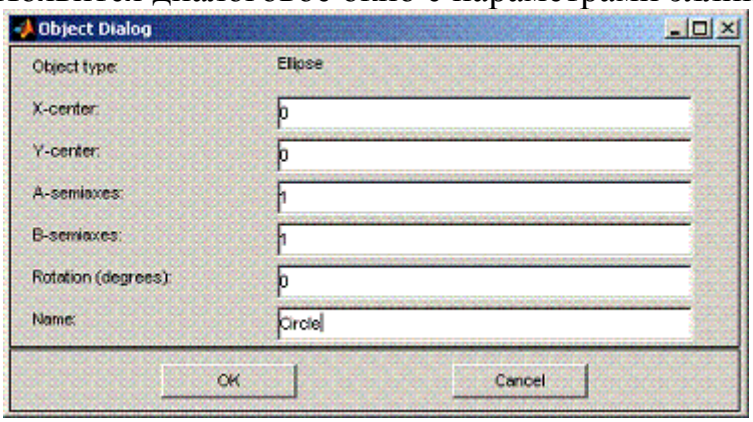


Рис. 2

Исправьте параметры так, чтобы получился единичный круг.

2. **Задание граничного условия**. Нажмите кнопку $\partial\Omega$. Граница круга выделится красным. Это означает, что на ней заданы условия Дирихле. Для их уточнения, зайдите в меню `Boundary` и выберете пункт `Specify Boundary Conditions`. Перед вами появится окно задания граничных условий.

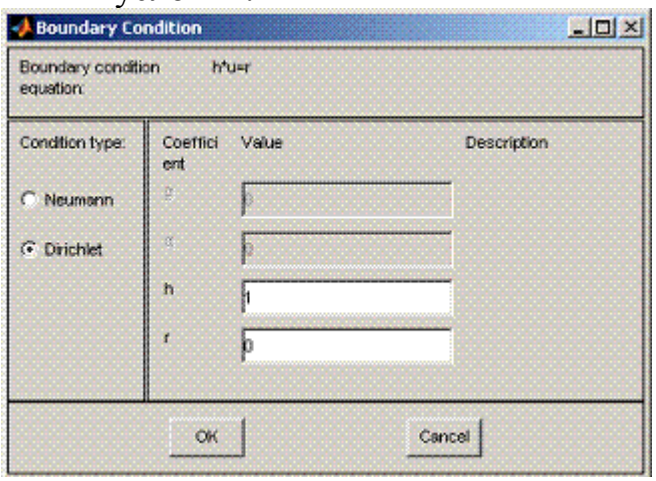


Рис. 3

В этом окне переключатель `Dirichlet`, указывает на задание граничных условий Дирихле. Исправьте в строке `r` значение 0 на 1. Тем самым, мы задаем граничное условие $u\big|_{\partial\Omega} = 1$.

3. **Задание конечно элементной сетки**. Триангуляция области может быть выполнена автоматически. Нажмите кнопку  . Это приводит к отображению используемой триангуляции области.

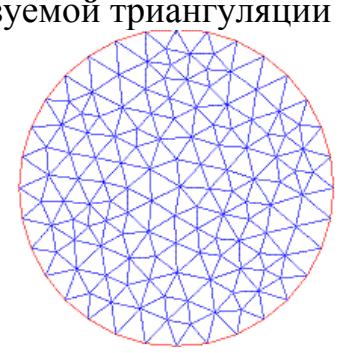


Рис. 4

Кнопка рядом  позволяет сгустить разбиение и, тем самым, повысить точность вычислений. Меню `Mesh` позволяет внести и другие изменения в конечноэлементную сетку. В нашем примере можно согласиться с разбиением области по умолчанию, и сетку не менять.

4. **Задание уравнения**. Зайдите в меню `PDE` и выберете пункт `PDE Specification`. Перед вами появится окно выбора вида ДУЧП. По умолчанию решается уравнение $-\Delta u = 10$. Исправьте в строке `f` значение 10 на $-16\cdot(x^2+y^2)$, т.е. введите это выражение, используя поэлементные знаки математических операций `-16*(x.^2+y.^2)`

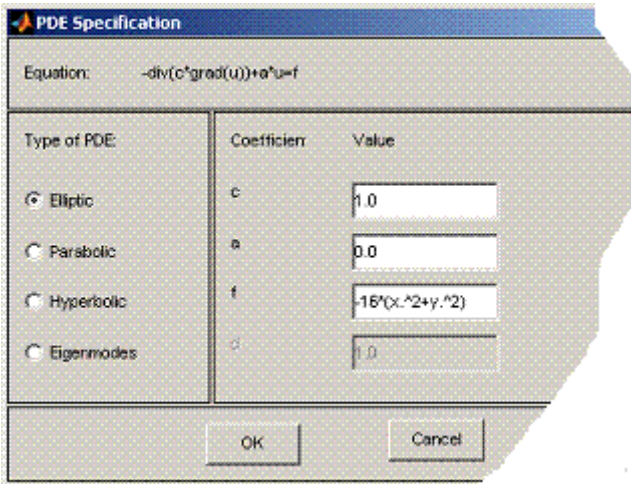
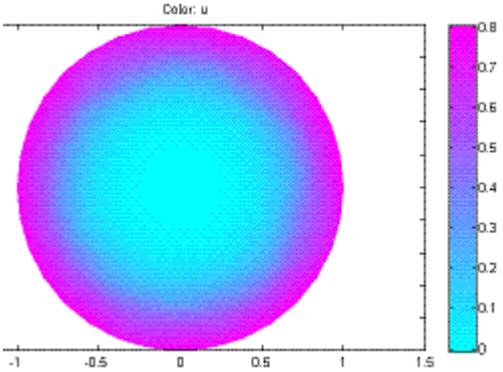


Рис. 5

Задав условия задачи, нажмите кнопку = для получения решения. В открытом окне вы получите цветной график приближенного решения, на котором цвет точек соответствует высоте (значению) решения *u(x,y)*. Палитра цветов отображается справа от рисунка.



Сравним полученное решение с точным $u = \left(x^2 + y^2\right)^2$, построив в области их разность. Для этого нажмите кнопку, на которой изображен логотип MatLab . Перед вами появится окно `Plot Selection`

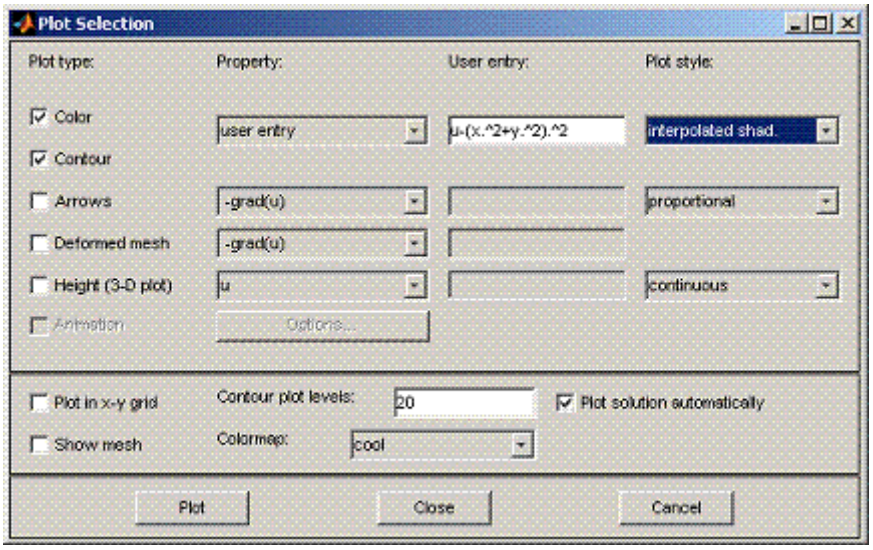
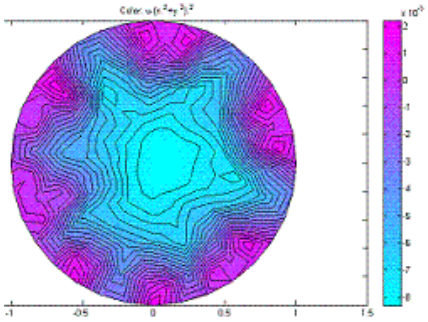


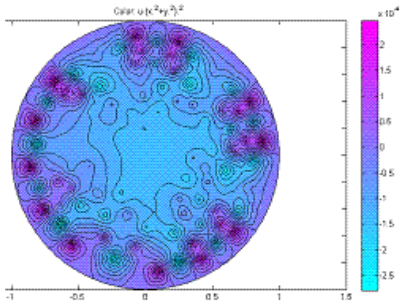
Рис. 6

Левая колонка этого окна содержит флаги, соответствующие способу визуализации результатов. Столбец полей `Property` состоит из

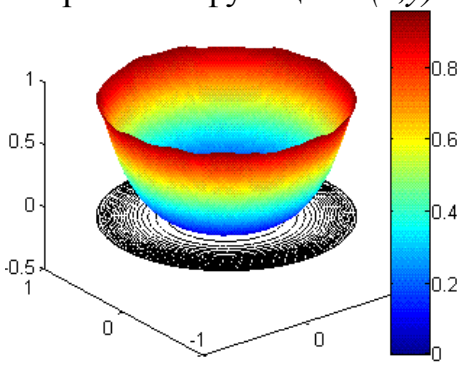
раскрывающихся списков, предназначенных для выбора отображаемой функции. Зайдите в верхний список, соответствующий флагам `Color` и `Contour` и в списке выберете пункта `User Entry`. Тогда станет доступной соответствующая ячейка третьей колонки. Введите в нее выражение `u-(x.^2+y.^2).^2`. Установите также флажок `Contour`. Затем нажмите кнопку `Plot` для построения графика погрешности решения $u-\left(x^2+y^2\right)^2$.



Из цветовой палитры графика вы можете определить значение максимальной погрешности (разности приближенного и точного решений). Кнопка  позволяет уменьшить размеры треугольников разбиения области, а следовательно, повысить точность решения краевой задачи. Нажмите на нее, а затем решите задачу еще раз, нажав кнопку  . На следующем рисунке показана погрешность решения, которая, как мы видим, стала значительно меньше.



Вы можете также увидеть график решения в виде поверхности. Нажмите кнопку  и установите в окне `Plot Selection` флаг `Height (3-D plot)`, а также в колонке полей `Property` напротив флага `Color` верните значение `u`. Затем нажмите кнопку `Plot`, которая создаст графическое окно `Figure 1` с графиком поверхности функции *u(x,y)*.



Пример 2. Решим задачу о стационарном распределении температуры в прямоугольной области с круговым отверстием в центре. Левая Γ_1 и правая

Γ_3 границы прямоугольной области теплоизолированы. На верхней Γ_2, нижней Γ_4 и внутренней Γ_5 границе области поддерживается постоянная температура (разная на разных участках границы). Источников тепла нет.

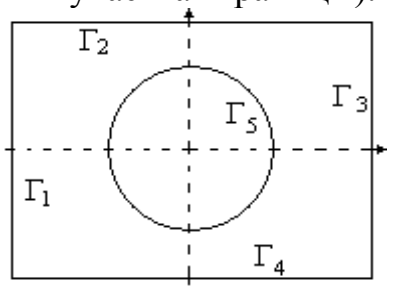


Рис. 7

Стационарное распределение температуры описывается дифференциальным уравнением

$$\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}=0 \qquad \text{или} \quad div(grad\, u)=0$$

и граничными условиями $u\big|_{\Gamma_2}=u_2$, $u\big|_{\Gamma_4}=u_4$, $u\big|_{\Gamma_5}=u_5$, $\left.\frac{\partial u}{\partial x}\right|_{\Gamma_1}=0$, $\left.\frac{\partial u}{\partial x}\right|_{\Gamma_3}=0$.

Задание двумерной области Ω. Конструирование области, показанной на рисунке, состоит в создании прямоугольника и круга заданных размеров, и вычитание области круга из области прямоугольника.

Нарисуем мышью прямоугольник и круг. В окне свойств прямоугольника зададим координаты левого нижнего угла (-3, -2), а также ширину 6 и высоту 4 прямоугольника. Содержимое R1 поля `name` оставим без изменений. Аналогично в окне свойств эллипса введем координаты центра (0, 0) и одинаковые значения полуосей 1. Содержимое E1 поля `name` оставим без изменения.

В данный момент установлен режим рисования. Переход в него происходит при использовании инструментов рисования или выборе меню `Draw – Draw Mode`. Другие режимы задаются выбором пунктов меню `Boundary - Boundary Mode` (режим задания граничных условий), `PDE - PDE Mode` (режим выбора уравнения) и `Mesh - Mesh Mode` (режим построения сетки), или выбором подходящих инструментов и пунктов меню.

Следующий этап состоит в определении взаимосвязи между примитивами, образующими область. Связь между ними определяется в строке `Set Formula`. Знак плюс означает объединение объектов, а минус – вычитание. Нашей области соответствует формула `R1 - C1`.

Выбор уравнения и задание граничного условия. Меню `Options - Application` позволяет задать тип решаемой задачи. Его пункт `Heat Transfer` соответствует задаче о распределении тепла. Выберите этот пункт. Слева от названия появится флаг – среда pdetool теперь настроена на решение задачи теплопроводности. Использование раскрывающегося списка,

размещенного на панели инструментов, приводит к аналогичному результату (на рис.1 в этом поле стоит выбор Generic Scalar). Теперь установим режим задания дифференциального уравнения, выбрав пункт PDE – PDE Mode. На экране отобразится область с отверстием. Выберите пункт PDE – PDE Specification... или выполните двойной щелчок по области. Появляется диалоговое окно PDE Specification (см. рис. 5, но с другими настройками). В верхней части окна на панели Equation содержится общий вид стационарного уравнения теплопроводности $-div(k \cdot grad\, T) = Q + h \cdot (Text - T)$, которое может быть решено в среде pdetool.

Значения коэффициентов устанавливаются в строках ввода, расположенных в средней части диалогового окна. Левая панель служит для выбора типа уравнения. Переключатель Elliptic соответствует задаче о стационарном распределении тепла, а Parabolic – нестационарному случаю. Установите переключатель Elliptic и задайте в строках ввода коэффициенты уравнения рассматриваемой задачи: *k=1, Q=0, h=0, Text=0* (физический смысл этих коэффициентов обсуждается в курсах математической физики, а краткое пояснение находится справа от соответствующего поля). Нажмите ОК для сохранения проделанных изменений.

Переходим к заданию граничных условий. Выберите пункт меню Boundary – Boundary Mode. Среда pdetool перейдет в режим установки граничных условий, и в ее окне будут отображены только границы области. Обратите внимание, что прямоугольник имеет четыре границы, по числу сторон, и окружность также составлена из четырех дуг. Щелчком мыши сделайте текущей верхнюю границу прямоугольника и выберите в меню Boundary пункт Specify Boundary Conditions... (или выполните двойной щелчок по желаемому участку границы). Появится диалоговое окно Boundary Condition, предназначенное для задания граничного условия на выбранном участке границы (такое как на рис. 3). На верхней границе прямоугольника мы задаем температуру. Это граничное условие Дирихле. Убедитесь, что на панели Condition type включен переключатель Dirichlet и в полях значений параметров установите *h=1, r=100*. Сохраните значения, нажав ОК, и проделайте аналогичную операцию для нижней стороны прямоугольника, предварительно сделав ее текущей.

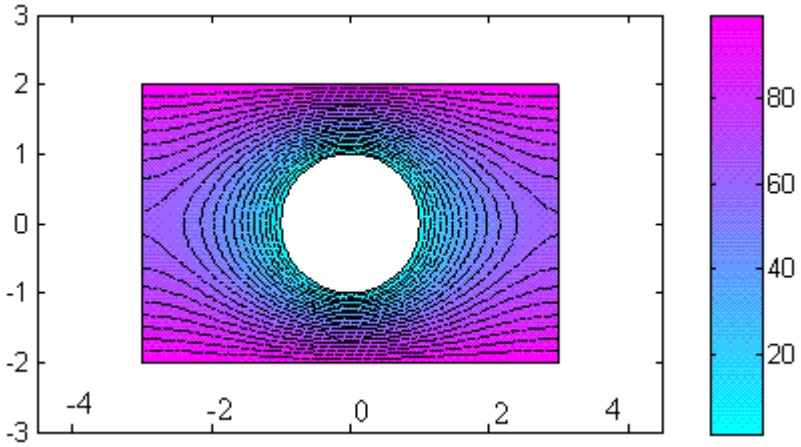
Граничные условия можно задавать по отдельности на каждой части границы, или объединить несколько частей и определить одинаковые граничные условия сразу для всей группы. Добавление части границы в группу производится щелчком мыши с одновременным удержанием клавиши Shift.

Установите нулевую температуру на границе отверстия, сгруппировав предварительно четыре части окружности. На правой и левой стороне прямоугольной области поток тепла равен нулю. Равенство нулю потока есть

частный случай условия Неймана. Установите переключатель `Condition type` в положение `Neumann`. Для задания теплоизолированных границ нужно задать *g=0, q=0*. В режиме установки граничных условий границы с условиями Дирихле отображаются красным цветом, а синий цвет выделяет границы, на которых задано условие Неймана.

Задание разбиения области. Выполним триангуляцию области – покрытие области сеткой, состоящей из треугольников. Триангуляция и установка ее параметров производятся в меню `Mesh`. Перейдите в режим триангуляции, выбрав пункт `Mesh Mode`. Область разбивается на достаточно крупные треугольные элементы, причем считается, что граница области составлена из сторон некоторых элементов. Инициализация и включение режима триангуляции может быть произведена кнопкой с треугольником  . Для получения решения с приемлемой точностью начальной триангуляции недостаточно, следует уменьшить шаг разбиения области. Выберите пункт `Refine Mesh` или нажмите кнопку  с треугольником, разделенным на четыре части. Каждый выбор данного пункта приводит к уменьшению размеров треугольников. Учтите, что выбор слишком мелкой сетки может привести к значительным затратам времени на решение задачи, однако слишком грубая сетка приводит к большим погрешностям. Вернуться к начальной триангуляции можно с помощью меню `Initialize Mesh` или кнопки с треугольником. Обратите также внимание на то, что уменьшение размеров ячеек сетки улучшает вид границ области.

Решение уравнения. Решение задачи производится выбором пункта `Solve - Solve PDE` или нажатием на кнопку со знаком равно  . Найденное распределение температуры отображается в окне среды pdetool контурным графиком с цветовой заливкой, рядом с которым расположен столбик с информацией о соответствии цвета значению температуры.



Изменение геометрии области, граничных условий, типа уравнения и его коэффициентов может быть выполнено даже после нахождения решения. Для этого следует перевести среду pdetool в соответствующий режим и произвести необходимые действия.

Сохранение работы производится в m-файле из пункта `Save as...` или `Save` меню `File`. Этот файл содержит функции PDE Toolbox, с помощью которых решается задача. Он содержит информацию о геометрии области, о типе и значениях коэффициентов уравнения и граничных условий, а также текущие установки среды. Впоследствии, загрузив этот файл, можно продолжить решение задачи.

Пример 3. Решим задачу о нестационарном распределении тепла в области, изображенной на рис. 7. На внешних границах прямоугольника поддерживается постоянная температура 1, а края отверстия подвергаются нагреву, температура изменяется одинаково во всех точках окружности линейно со временем. Внутри области нет источников тепла, в начальный момент времени температура равна нулю во всей области.

Задайте геометрию области и настройте среду pdetool на решение задачи теплопроводности. Перейдите к диалоговому окну `PDE Specification` и выберите параболический тип уравнения, установив переключатель `Parabolic`. Задайте следующие значения коэффициентов *rho=1, C=1, k=1, Q=0, h=0, Text=0*. На границах прямоугольника задайте единичную температуру. Для окружности в поле `r` введите формулу *t*, где переменная `t` используется для обозначения времени, и задайте *h=1*.

Задайте распределение температуры в начальный момент времени, и интервал времени, в который следует найти приближенное решение. Для этого выберите пункт `Solve - Parameters...` Появляется диалоговое окно `Solve Parameters` (рис. 8), вид которого соответствует типу решаемой задачи. Установите в строке `Time` вектор моментов времени 0:0.05:1, а в строке `u(t0)` задайте нулевое начальное распределение температуры, которое в общем случае может зависеть от *x* и *y*.

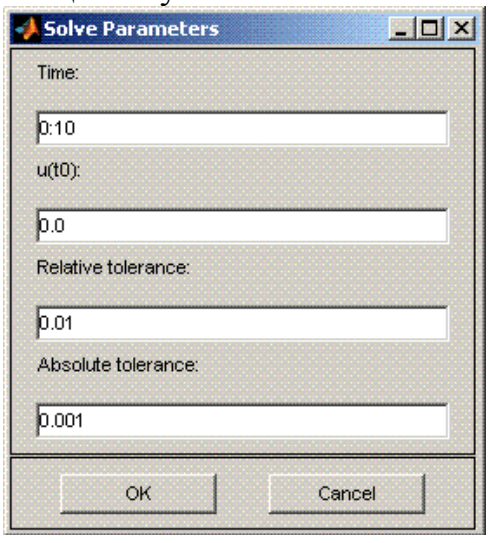


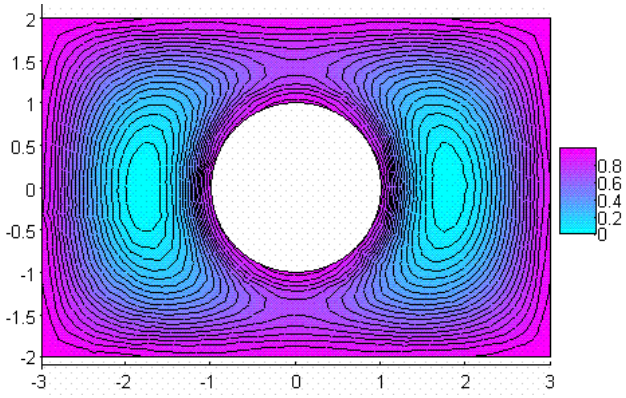
Рис. 8

Если потребуется найти решение не в равноотстоящие моменты времени, то в поле `u(t0)` можно задать вектор значений, например, [0 0.01 0.1 0.2 0.5 0.75 1] или можно использовать одну из функций MatLab генерирующую вектор.

Инициализируйте триангуляцию, уменьшите шаг сетки в несколько раз и решите задачу. Решение займет некоторое время, которое зависит от производительности компьютера. Процесс решения сопровождается выводом информации в командное окно.

```
53 successful steps
0 failed attempts
108 function evaluations
1 partial derivatives
13 LU decompositions
107 solutions of linear systems
```

В окне pdetool появляется распределение температуры в области в конечный момент времени



Для того чтобы проследить за динамикой процесса, следует установить в диалоговом окне `Plot Selection` флаг `Animation` и воспользоваться кнопкой `Options` для определения параметров анимации. Откроется окно `Animation Options`, изображенное на следующем рисунке

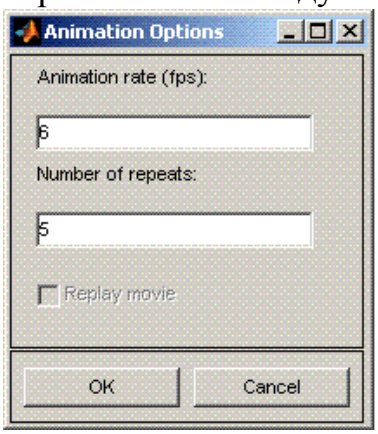


Рис. 9

Строка `Animation rate (fps)` служит для задания числа кадров в секунду (frames per second). Число повторов устанавливается в поле `Number of repeats`. Введите желаемые значения или оставьте те, что используются по умолчанию. Закройте окно кнопкой OK и нажмите кнопку `Plot` в окне `Plot Selection`. Динамическое распределение температуры в области отображается в отдельном графическом окне.

Чтобы посмотреть отдельные кадры анимации их следует сохранить в массив. Для этого служит меню `Plot - Export Movie`. Открывается окно `Export`, в котором следует ввести имя массива, в элементах которого будут храниться кадры анимации. Например, введем имя MyMovie. Закроем

окно кнопкой ОК, в окне `Animation options` установим число повторов равное единице и еще раз решим задачу. Во время решения все кадры будут сохранены в наш массив.

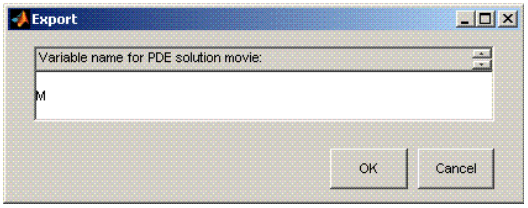


Рис. 10

Теперь мы можем преобразовать любой кадр, например четвертый, в массив данных командой

[X,Map] = frame2im(MyMovie(4));

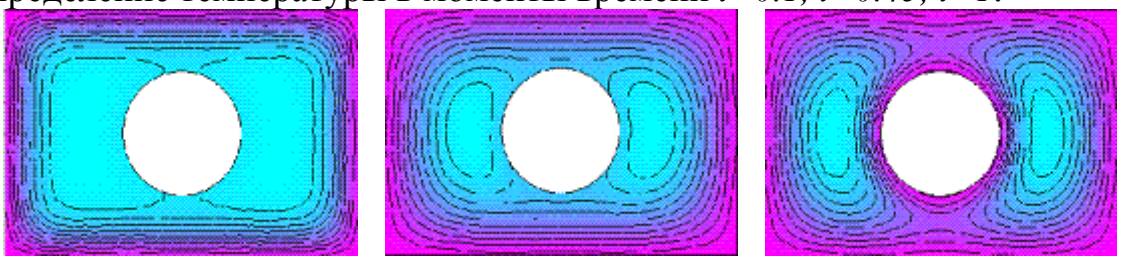
Функция **frame2im** преобразует единичный кадр анимации в числовой массив X и возвращает массив палитры цветов `Map`. Чтобы отобразить массив `X` в графическом окне мы можем выполнить команду

image(X)

Последовательно выполняя команды

[X,Map] = frame2im(MyMovie(5)); image(X)

с переменным номером кадров, вы можете построить (и сохранить в графический файл) любой кадр анимации. На следующем рисунке показано распределение температуры в моменты времени *t=0.1, t=0.45, t=1*.



Системы дифференциальных уравнений

Функции, входящие в состав PDE Toolbox, позволяют решить систему дифференциальных уравнений произвольной размерности. Среда pdetool оперирует только с системой второго порядка:

$$-\nabla\cdot\left(c_{11}\nabla u_1\right)-\nabla\cdot\left(c_{12}\nabla u_2\right)+a_{11}u_1+a_{12}u_2=f_1;$$
$$-\nabla\cdot\left(c_{21}\nabla u_1\right)-\nabla\cdot\left(c_{22}\nabla u_2\right)+a_{21}u_1+a_{22}u_2=f_2;$$

Ее решением является вектор-функция $\left(u_1(x,y),u_2(x,y)\right)$. Для однозначного определения решения необходимо задать граничные условия. На различных частях границы можно задавать условия Дирихле

$$h_{11}u_1+h_{12}u_2=r_1$$
$$h_{21}u_1+h_{22}u_2=r_2$$

Неймана

$$\mathbf{n}\cdot\left(\begin{pmatrix}c_{11} & c_{12}\\ c_{21} & c_{22}\end{pmatrix}\nabla u\right)+\begin{pmatrix}q_{11} & q_{12}\\ q_{21} & q_{22}\end{pmatrix}\begin{pmatrix}u_1\\ u_2\end{pmatrix}=\begin{pmatrix}g_1\\ g_2\end{pmatrix}$$

или смешанные граничные условия более сложного вида (см. справку).

Выбор опции `Generic System` в подменю `Application` меню `Options` (или в раскрывающемся списке на панели инструментов) настраивает среду pdetool на решение подобной системы уравнений. Диалоговое окно `PDE Specification` позволяет задать коэффициенты системы уравнений (рис. 11).

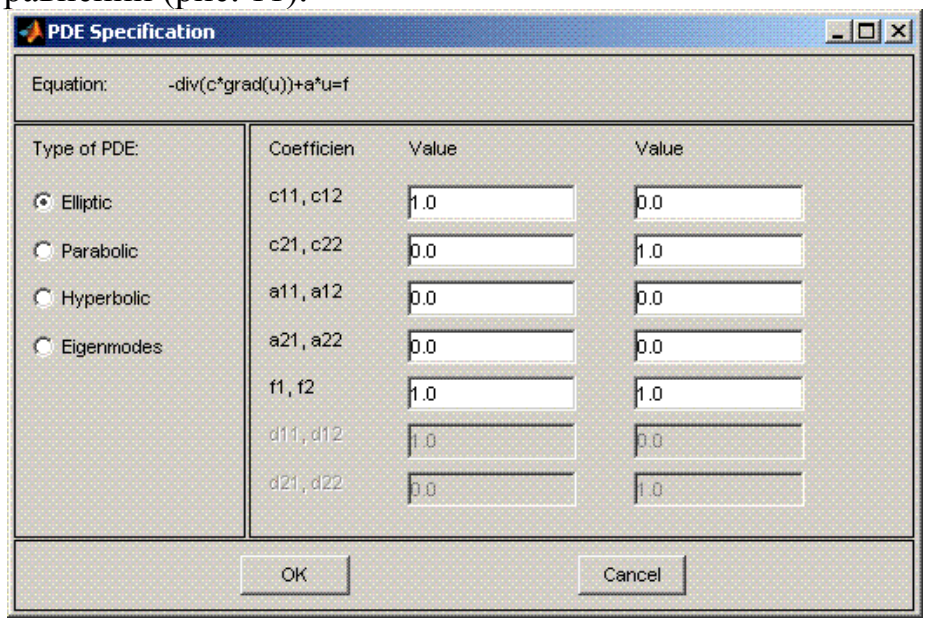


Рис. 11

В первом столбце `Value` этого окна сверху вниз задаются коэффициенты `c11, c21, a11, a21, f1`, а во втором – оставшиеся коэффициенты. Как видим, можно решать не только эллиптические (стационарные) системы, но и нестационарные (гиперболические и параболические), а также задачи на собственные значения.

Триангуляция двумерной области выполняется аналогично тому, как это делается для одного уравнения.

Пример 4. Решим систему дифференциальных уравнений в частных производных второго порядка

$$\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}=-q_0$$

$$\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}=-\frac{u}{D}$$

в единичном круге с граничными условиям $u\big|_{\partial\Omega}=0, v\big|_{\partial\Omega}=0$.

Вначале в среде pdetool выберем тип решаемой задачи – `Generic System` и откроем окно `PDE Specification` (рис. 11), в котором следует задать коэффициенты системы. В нашем случае для первого уравнения следует задать `c11=1, c12=0, a11=0, a12=0, f1=1`. Для второго уравнения задаем `c21=0, c22=1, a21=-1, a22=0, f2=0`. Эти значения соответствуют системе с q_0=1 и D=1.

Окно `Boundary Condition` (рис. 12) содержит переключатели `Neumann`, `Dirichlet` и `Mixed` для выбора одного из трех типов условий, перечисленных выше, и строки ввода для задания коэффициентов граничных условий. Нулевым значениям искомых функций на границе соответствуют условия Дирихле с параметрами **h11=1, h12=0, r1=0 и h21=0, h22=1, r2=0**.

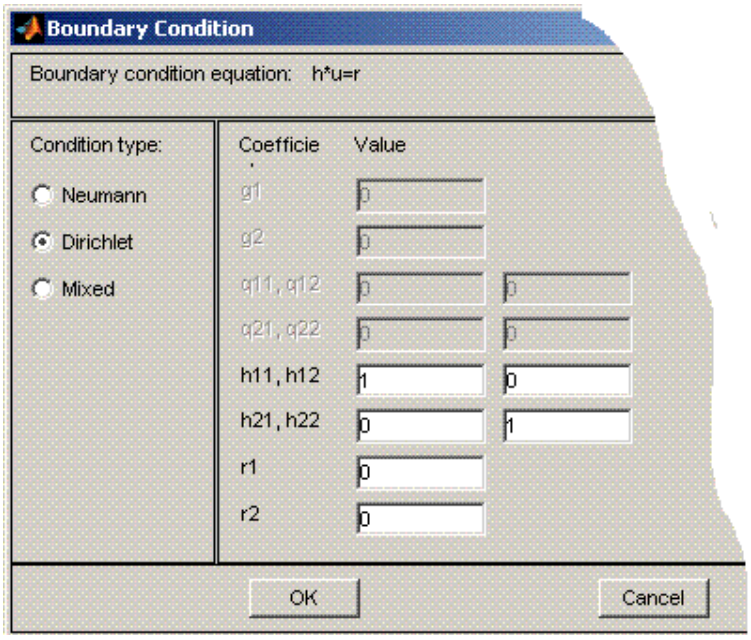
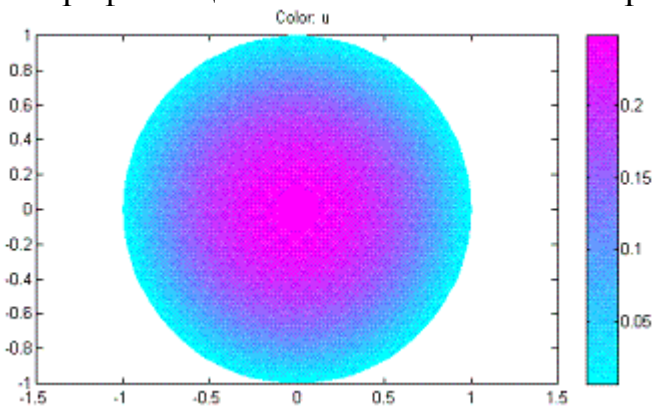
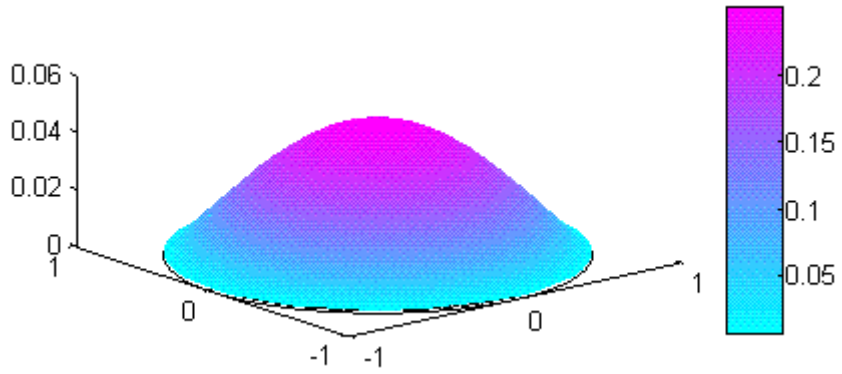


Рис. 12

Задаем, как обычно, триангуляцию области и решаем задачу. В окне pdetool получаем контурный график с цветовой заливкой и палитрой цветов справа.



По умолчанию, строится график первой функции (в нашем случае $u(x,y)$). Диалоговое окно `Plot Selection`, открываемое меню `Plot – Parameters`, предоставляют возможность визуализировать любую (первую или вторую) компоненты решения (они обозначены через **u** и **v**). Установите в этом окне флажок `Height (3-D plot)`, в поле напротив него в столбце `Property` выберите **v** и нажмите кнопку `Plot`. В графическом окне будет построен график функции $v(x,y)$.



Для нашей задачи известно точное решение. Оно имеет вид

$$u(x,y)=\frac{q_0}{4}\left(1-x^2-y^2\right)$$

$$v(x,y)=\frac{q_0}{64D}\left(x^4+2x^2y^2+y^4-4x^2-4y^2+3\right)=$$

$$=\frac{q_0}{64D}\left(x^2+y^2-1\right)\left(x^2+y^2-3\right)$$

Действительно, $u''_{xx}=-\frac{q_0}{2}, u''_{yy}=-\frac{q_0}{2}, u''_{xx}+u''_{yy}=-q_0$. Далее

$$v''_{xx}=\frac{q_0}{64D}\left(12x^2+4y^2-8\right),\quad v''_{yy}=\frac{q_0}{64D}\left(12y^2+4x^2-8\right)$$

и

$$v''_{xx}+v''_{yy}=\frac{q_0}{4D}\left(x^2+y^2-1\right)=-\frac{u}{D}.$$

Очевидно, что оба граничных условия удовлетворены.

Чтобы проверить полученное в pdetool приближенное решение создадим функцию, выполняющую вычисления функции *v(x,y)* по приведенной выше формуле

```
function w=CirclePlateSettlement(x,y)
% точное решение 1 ДУЧП примера 4
q0=1;
D=1;
w=q0/64/D*(x.^4 + 2.*x.^2.*y.^2+y.^4-4.*x.^2-4.*y.^2+3);
```

Теперь мы можем построить контурный график разности функции $v(x,y)$ и точного решения. Для этого откроем окно `Plot Selection` (рис. 13) и в первом поле столбца `Property` выберем `user entry`, а в первом поле столбца `user entry` введем формулу `v- CirclePlateSettlement(x,y)` и нажмем кнопку `Plot`.

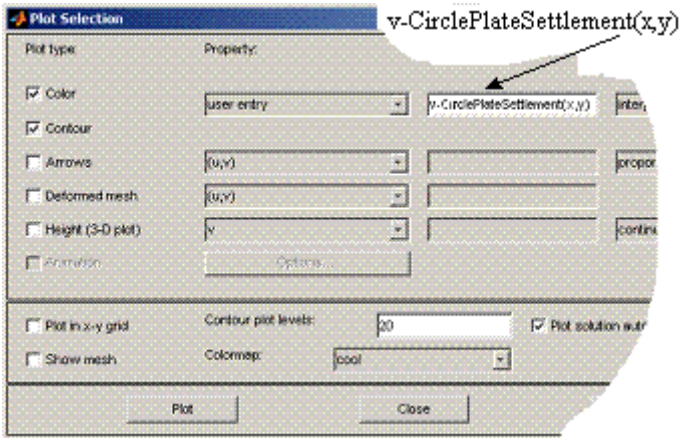
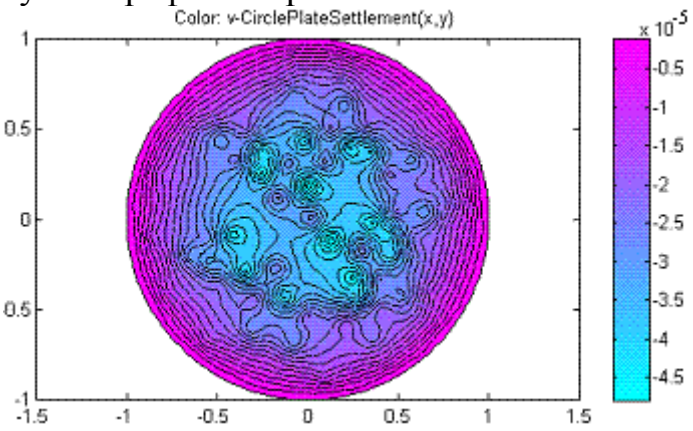


Рис. 13

В окне pdetool получим график погрешности



Числовой множитель 10^{-5} вверху палитры показывает, что погрешность весьма мала. Сохраним наше решение в файле `CirclePlatePde.m`

Нелинейные уравнения.

В среде `pdetool` можно решать нелинейные дифференциальные уравнения в частных производных

Пример 5. Минимальные поверхности. Решим ДУЧП следующего вида

$$-\nabla\cdot\left(\frac{1}{\sqrt{1+|\nabla u|^2}}\nabla u\right)=0$$

в единичном круге $\Omega=\{(x,y):\ x^2+y^2\le 1\}$ со значениями $u=x^2$ на границе $\partial\Omega$. Оно определяет форму минимальной поверхности, край которой находится на заданной высоте над границей круга.

1. Вначале укажем тип решаемого уравнения `Generic Scalar`.
2. Нарисуем единичный круг так, как мы это делали раньше.
3. Перейдем в режим задания граничных условий. Выполним команду меню `Edit - Select All` для выбора всех участков границы круга. Выполним двойной щелчок по контуру границы и зададим условие Дирихле в поле `r` в виде `x.^2`.
4. Уравнение в частных производных, описывающее минимальную поверхность, имеет вид

$$div\left(\frac{grad\,u}{\sqrt{1+\left(u'_x\right)^2+\left(u'_y\right)^2}}\right)=0$$

Откроем диалоговое окно `PDE Specification` и в поле **c** введем выражение `1./sqrt(1+ux.^2+uy.^2)`, а в полях `a` и `f` введем ноль. Проверьте, что установлен переключатель типа уравнения `Elliptic`.

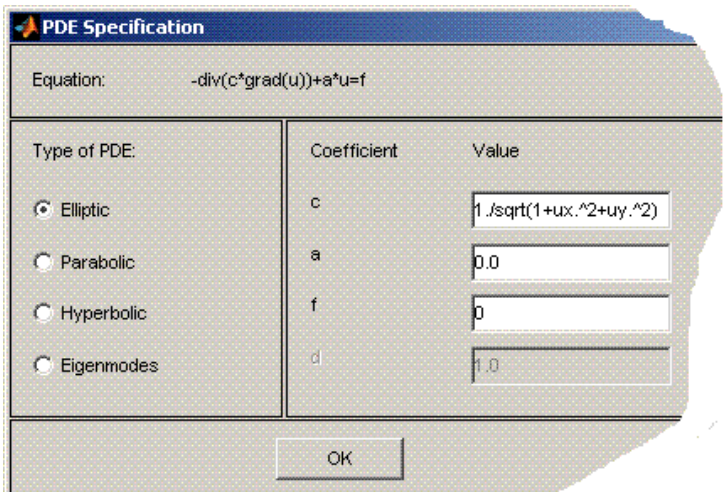


Рис. 14

5. На следующем шаге задайте триангуляцию

6. Перед тем как решать уравнение, выберите меню `Solve - Parameters` и в открывшемся окне установите флажок `Use nonlinear solver`,

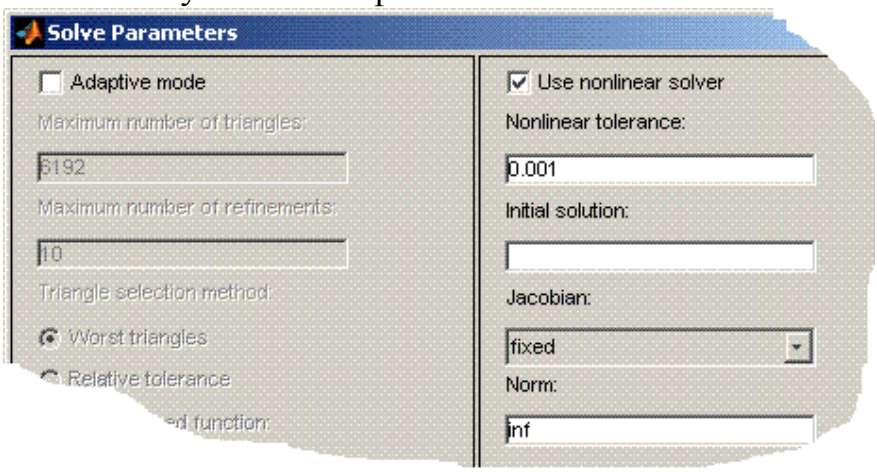
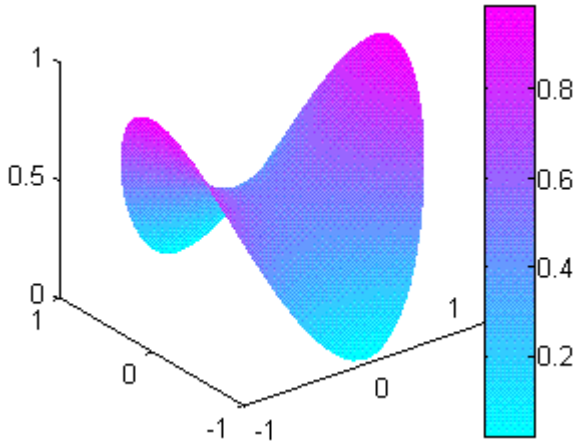


Рис. 15

а в поле `Nonlinear tolerance` введите 0.001.

7. Теперь решим уравнение. В диалоговом окне `Plot Selection`, установите флажок `Height (3-D plot)` и нажмите кнопку `Plot` для построения графика поверхности



Сохраним решение pdetool в файле `pdeminsurf.m`

4.1.2 Решение ДУЧП с помощью функций PDE TOOLBOX

Среда pdetool с графическим интерфейсом пользователя предназначена для облегчения доступа к ядру PDE Toolbox – набору функций, реализующих основные этапы решения задачи: задания и разбиения области, задание уравнения, граничных условий и визуализации результата.

Решение модельных задач

Пример 1. Использование функций PDE Toolbox рассмотрим на примере решения эллиптического уравнения

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = -20 \cdot \sin \pi x \cdot \sin 2\pi y$$

в прямоугольнике с центром в начале координат, размерами 2×1 и нулевыми граничными условиями $u\big|_{\Gamma} = 0$.

Последовательность решения задачи такова:

- задание геометрии области и сохранение ее в переменных рабочей среды;
- задание граничных условий и сохранение их в переменных рабочей среды;
- триангуляция области и сохранение результатов в переменных рабочей среды;
- присваивание некоторым переменным значений коэффициентов решаемого уравнения и создание строковых выражений функций, используемых в уравнении;
- вызов функции – решателя ДУЧП (для разных типов уравнений и систем они разные) с передачей ей аргументов, содержащих геометрию среды, граничные условия, информацию о триангуляции, а также значения коэффициентов и функций – членов уравнения; сохранение результата в переменную среды;

– вызов функции визуализации, аргументами которой являются переменные, содержащие информацию о триангуляции и вектор результата.

Разберем последовательно шаги решения задачи, описанные выше. При этом первые два шага пока выполним в среде pdetool с последующим экспортом информации в переменные рабочей среды.

Настроим среду pdetool на решение скалярного эллиптического уравнения (Рис. 16),

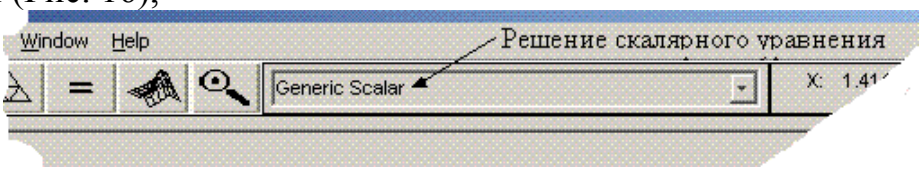


Рис. 16

установив переключатель `Elliptic` в окне `PDE Specification` (см. рис. 14).

Нарисуем в среде pdetool прямоугольник с центром в начале координат шириной два и высотой один и зададим нулевые граничные условия. Затем экспортируем информацию в переменные рабочей среды. Для этого выполним команду меню `Boundary` – `Export Decomposed Geometry, Boundary Cond's`.... Откроется диалоговое окно `Export`, в котором по умолчанию стоят имена `g` и `b` – первое для массива, хранящего информацию о геометрии границы, и второе – для хранения матрицы граничных условий.

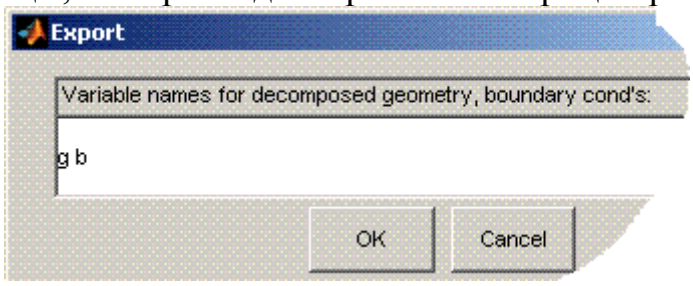


Рис. 17

Нажмем ОК, и в рабочей среде появятся двумерные массивы `g` и `b`. Посмотрим их содержимое

```
>>g
g =
    2.0000    2.0000    2.0000    2.0000
    1.0000   -1.0000   -1.0000    1.0000
   -1.0000   -1.0000    1.0000    1.0000
   -0.5000   -0.5000    0.5000    0.5000
   -0.5000    0.5000    0.5000   -0.5000
         0         0         0         0
    1.0000    1.0000    1.0000    1.0000
```

Двумерный массив `g` называется матрицей декомпозиционной геометрии (Decomposed Geometry matrix). Она хранит информацию о частях границы области, которые могут быть отрезком, частью дуги окружности или эллипса. Каждому столбцу матрицы соответствует участок границы.

Прямоугольник представляется четырьмя отрезками и поэтому в матрице 4 столбца. Двойка в первом элементе столбца означает, что столбец соответствует отрезку, второй и третий элементы содержат абсциссы начала

и конца отрезка, а четвертый и пятый — ординаты. Функции пакета pdetool предполагают, что вся область разбита на некоторые непересекающиеся подобласти, имеющие собственные идентификаторы (номера). Идентификаторы подобластей, расположенных слева и справа от элемента границы (отрезка или дуги), записаны в шестом и седьмом элементе. В нашем примере есть только одна область – прямоугольник (идентификатор единица). Слева от отрезков, представляющих границу прямоугольника, областей нет – идентификатор ноль.

Двумерный массив `b` называется матрицей граничных условий. Каждый ее столбец соответствует части границы. В нашем случае область является прямоугольником с одинаковыми нулевыми граничными условиями Дирихле, и матрица состоит из четырех столбцов.

>>b

```
b =
     1     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1
    48    48    48    48
    48    48    48    48
    49    49    49    49
    48    48    48    48
```

Столбец условно делится на две части, верхняя часть столбца матрицы граничных условий содержит информацию о типе граничных условий и длинах формул. В нижней части записаны коэффициенты и формулы граничных условий, причем каждый символ строки с формулой хранится в последовательно идущих элементах, содержащих коды символов. В случае одного уравнения и граничного условия Дирихле коды символов формулы хранятся в элементах столбца, начиная с седьмого. Преобразуем их в символы с помощью функции char

char(b(7:end,:))

```
ans =
0000
0000
1111
0000
```

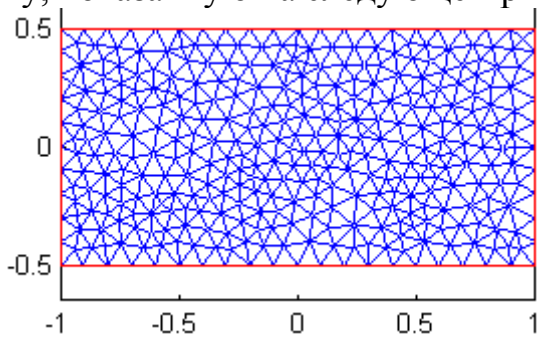
Первые два элемента зарезервированы под коэффициенты условия Неймана (они равны нулю, поскольку поставлено условие Дирихле). Третий элемент содержит значение `h` (в нашем случае единица), а остальные являются символами правой части выражения `h*u=r` (в нашем случае ноль).

Триангуляция области выполняется функцией **[p, e, t] = initmesh(g)**, которая разбивает плоскую область на треугольники. Ее входным аргументом является матрица декомпозиционной геометрии. Три выходных аргумента (матрицы) **p, e, t** содержат информацию о триангуляции (их структуру мы рассмотрим позже). Функция **initmesh** может содержать дополнительные аргументы, задаваемые парами: название свойства, значение. Например, свойство `Hmax` устанавливает максимально допустимое

значение длины стороны треугольного элемента. Полученную сетку можно отобразить в отдельном окне при помощи функции **pdemesh**, входными аргументами которой являются вышеперечисленные массивы. Например, последовательность команд

```
>>[p, e, t] = initmesh(g, 'Hmax', 0.1);
>> pdemesh(p,e,t); axis equal
```

создает и выводит сетку, показанную на следующем рисунке



На следующем шаге мы создаем переменные, которые будут коэффициентами уравнения $-div(c \cdot grad(u)) = f$. Если бы мы определяли уравнение в среде pdetool, то открыли бы окно `PDE Specification`

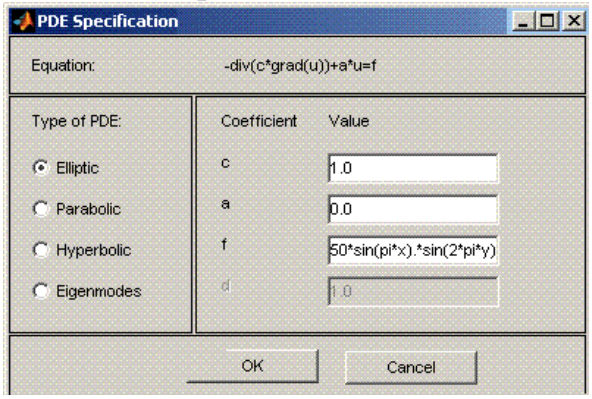


Рис. 18

в котором ввели бы значения коэффициентов `a` и `c` уравнения и выражение для функции `f` – правой части уравнения. Для работы с функциями пакета мы в рабочей среде создаем эти переменные, при этом выражение для функции создаем в строковом виде

```
>>a=0;                          % коэффициент уравнения
>>c=1;                          % коэффициент уравнения
>>f='20*sin(pi*x).*sin(2*pi*y)'; % Строка с формулой правой части уравнения
```

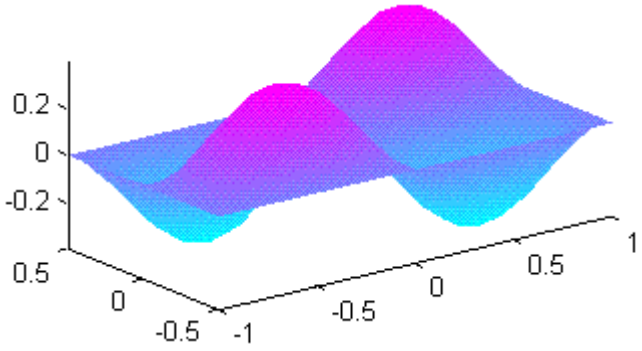
Теперь можно решать задачу. Для решения эллиптических уравнений используется функция **assempde**. Ей следует передать аргументы – матрицу граничных условий `b`, массивы `p, e, t,` содержащие информацию о триангуляции, и коэффициенты уравнения `c, a, f`.

```
>>u=assempde(b,p,e,t,c,a,f);
```

Функция **assempde** возвращает вектор решений – значения искомой функции в узлах сетки.

На последнем шаге можно построить график решения. Функция **pdesurf** рисует поверхность функции *u(x,y)*, используя информацию о триангуляции, хранящуюся в переменных `p` и `t` и вектор решения `u`.

>>pdesurf(p,t,u);



Решение приведенной задачи удобно организовать в форме сценария FuncPDE_1.m

```
% Сценарий решения первого ДУЧП функциями PDE Toolbox
% Предполагается, что матрица декомпозиционной геометрии g
% и матрица граничных условий b уже созданы.
% 1. Триангуляция с использованием массива декомпозиционной геометрии g
[p, e, t] = initmesh(g, 'Hmax', 0.1);
pdemesh(p,e,t);                    % построение сетки для проверки
% 2. Задание коэффициентов уравнения
a=0;
c=1;
% 3. Определение строки с формулой правой части уравнения
f='5*pi^2*sin(pi*x).*sin(2*pi*y)';
% 4. Решение эллиптического уравнения
u=assempde(b,p,e,t,c,a,f);    % используем матрицу граничных условий b
% 5. Построение графика поверхности решения
pdesurf(p,t,u);
axis equal;
```

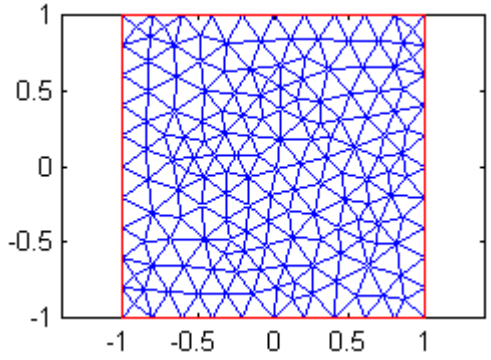
Меняя функцию правой части *f(x,y)*, мы можем для заданной формы области (прямоугольник размерами 2×1) и нулевых граничных условий $u\big|_{\Gamma}=0$ получать решения уравнения Пуассона. Если нужно изменить форму области или граничные условия, то следует изменить матрицу декомпозиционной геометрии `g` и матрицу граничных условий `b`. Их можно создавать, экспортируя из среды pdetool, или создавать вручную. Вместо этих матриц можно также использовать функции специальной структуры. О них мы поговорим позже.

Пример 2. Рассмотрим пример решения уравнения колебания квадратной мембраны, закрепленной на контуре. Эта задача отличается от предыдущей тем, что необходимо задать начальные условия и моменты времени в которые следует определять решение.

Нарисуем в среде pdetool квадрат с центром в начале координат со стороной 2 ($-1\le x,y\le1$) и зададим нулевые граничные условия $u\big|_{\Gamma}=0$. Затем экспортируем информацию в переменные рабочей среды с помощью команды меню `Boundary – Export Decomposed Geometry, Boundary Cond's...`. Назовем матрицу декомпозиционной геометрии `gw` и матрицу граничных условий `bw`.

Выполним триангуляцию области и построим сетку командами

```
>>[p, e, t] = initmesh(gw, 'Hmax', 0.1);
>> pdemesh(p,e,t); axis equal
```



Матрица `p`, которую возвращает функция **initmesh**, содержит две строки – *x* и *y* координаты узлов сетки.

```
>>p
p =
Columns 1 through 8
 1.0000   -1.0000   -1.0000    1.0000    0.8000    0.6000    0.4000
-1.0000   -1.0000    1.0000    1.0000   -1.0000   -1.0000   -1.0000

...
Column 177
 -0.3128
 -0.0361
```

Для экономии места мы не приводим все элементы вектора `p`. Создадим вектора, содержащие *x* и *y* координаты узлов.

```
x=p(1,:)';
y=p(2,:)';
```

Они нам нужны для задания начальных условий в узлах сетки.

```
u0=sin(pi*x).*sin(pi*y);   % начальное смещение точек мембраны
ut0= 0;                    % начальная скорость точек мембраны
```

Теперь определим моменты времени, в которые будем находить решение

```
n=26;
tlist=linspace(0,5,26)     % вектор моментов времени
tlist =
  Columns 1 through 8
         0    0.2000    0.4000    0.6000       ...
```

Для решения гиперболического уравнения используется функция **hyperbolic** пакета PDE Toolbox. Ее аргументами являются:

- вектора u_0, u_{t0} начальных значений (начальных смещений и начальных скоростей) в узлах сетки;
- вектор t_{list} моментов времени, в которые определяется решение;
- матрица граничных условий `bw`;
- вектора **p, e, t,** возвращаемые функцией **initmesh**, содержащие информацию о триангуляции;
- коэффициенты `c, a, f, d` уравнения $d\dfrac{\partial^2 u}{\partial t^2} - div(c \cdot grad(u)) + a \cdot u = f$, те же, что мы вводили бы в окне PDE Specification.

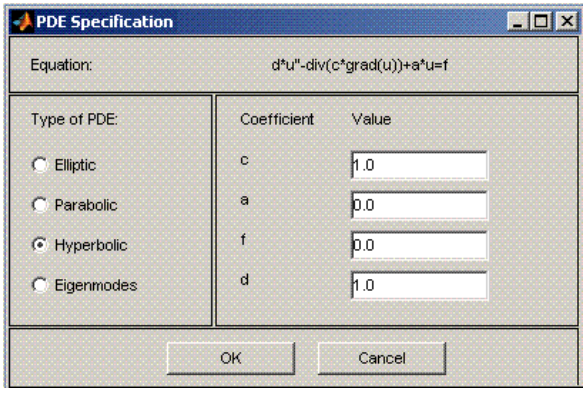


Рис. 19

В нашей задаче `c=1, a=0, f=0, d=1`. Выполним команду решения гиперболического уравнения

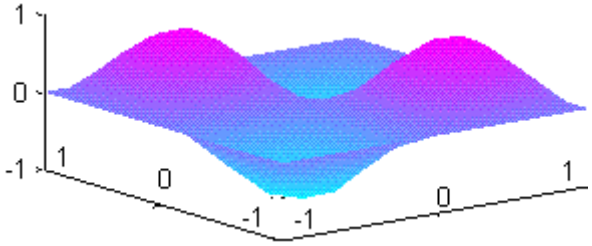
uw=hyperbolic(u0,ut0,tlist,bw,p,e,t,1,0,0,1);

Через некоторое время в командном окне появятся сообщения об успешном решении задачи

```
696 successful steps
70 failed attempts
1534 function evaluations
1 partial derivatives
180 LU decompositions
1533 solutions of linear systems
```

Функция **hyperbolic** возвращает решение в виде матрицы, каждый столбец которой представляет вектор смещения узлов сетки в некоторый момент времени. Например, для построения поверхности мембраны в 5-й момент времени (t_5=`tlist(5)=0.8`) можно использовать команду

pdesurf(p,t,uw(:,5));



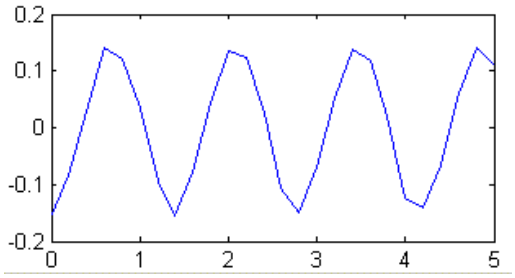
Наоборот, каждая строка матрицы `uw` представляет смещение некоторой точки (узла) в последовательные моменты времени. Например, 45 – й узел сетки имеет координаты

>> [x(45), y(45)]

```
ans =
    0.8710   -0.8741
```

График смещения точки во времени будет иметь вид

plot(tlist,uw(45,:));



Если бы сетка была более густой, а моменты времени, в которые мы определяем решение – короче, то предыдущий график получился бы более гладким. Функция **refinemesh** служит для уменьшения шага сетки. Первым входным аргументом является матрица декомпозиционной геометрии, затем передаются массивы **p, e, t** с информацией о триангуляции. В выходных аргументах возвращаются массивы, соответствующие измельченной сетке. По умолчанию сторона каждого элемента делится пополам, каждый треугольный элемент исходной сетки разбивается на четыре части. Команды, приведенные ниже, производят вышеописанное уменьшение шага сетки дважды

```
[p1,e1,t1]=refinemesh(gw,p,e,t);
[p2,e2,t2]=refinemesh(gw,p1,e1,t1);
pdemesh(p2,e2,t2); axis equal
```

Интересно, что 45 – й узел нового разбиения имеет те же координаты, что и 45 – й предыдущего разбиения

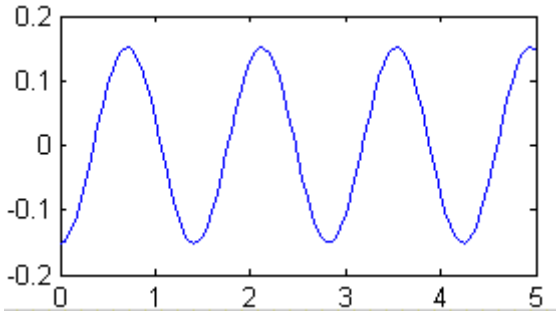
```
[p2(1,45),p2(2,45)]
ans =
    0.8710   -0.8741
```

Уплотним моменты времени вектора `tlist`, пересчитаем вектора начальных условий u_0, u_{t0} для узлов нового разбиения и снова решим задачу (это займет больше времени, чем предыдущее решение)

```
n=101; tlist=linspace(0,5,n);
x=p2(1,:)'; y=p2(2,:)';
u0=sin(pi*x).*sin(pi*y); ut0= 0;
uw=hyperbolic(u0,ut0,tlist,bw,p2,e2,t2,1,0,0,1);
```

График колебаний во времени 45 – го узла новой сетки, имеющего координаты (0.8710, -0.8741), будет иметь более гладкий вид

```
plot(tlist,uw(45,:));
```



Используя матрицу решения `uw` можно построить анимацию процесса колебания мембраны. В следующем файле – сценарии создается массив кадров анимации M, который затем проигрывается функцией **movie** (используем первый вектор моментов времени `tlist` и решение **uw=hyperbolic(u0,ut0,tlist,bw,p,e,t,1,0,0,1)**, полученное при первоначальном разбиении области)

```
% файл animw2.m
% сценарий создания анимации процесса колебания мембраны
% используется матрица uw решения в узлах сетки триангуляции,
% описываемой векторами p и t
umax=max(max(uw));            % максимальный элемент матрицы смещений
umin=min(min(uw));            % минимальный элемент матрицы смещений
```

```
newplot                        % создаем новое графическое окно
for i=1:n,
    pdesurf(p,t,uw(:,i));           % рисуем кадр – график поверхности
    axis([-1 1 -1 1 umin umax]);    % приводим кадр к общему масштабу
    M(i)=getframe;                  % запоминаем кадр в массив
end
movie(M,4);                         % проигрываем массив кадров 4 раза
```

Выполните команду

animw2

и в графическом окне будет показан процесс колебания мембраны. При этом массив кадров анимации `M` будет создан в рабочем пространстве. Чтобы еще раз посмотреть анимацию, можно, не запуская сценарий, выполнить команду

movie(M, кол-во повторений);

Заметим, что функция **max(A)**, где A – матрица, возвращает строку максимальных значений по столбцам A. Для того, чтобы найти максимальный элемент всей матрицы, мы в сценарии применяем дважды функцию **max**. Например

A=[1 7 3; -1 4 9; 5 0 3]

```
A =
     1     7     3
    -1     4     9
     5     0     3
```

max(A)

```
ans =
     5     7     9
```

max(max(A))

```
ans =
     9
```

Зададим начальное смещение мембраны в форме пирамиды и соберем все необходимые команды в один файл – сценарий `animw3.m`

```
% Сценарий решения уравнения колебаний мембраны функциями PDE Toolbox
% Предполагается, что матрица декомпозиционной геометрии gw
% и матрица граничных условий bw уже созданы в рабочем пространстве
%
% Триангуляция с использованием матрицы декомпозиционной геометрии gw
[p, e, t] = initmesh(gw, 'Hmax', 0.05);
% pdemesh(p,e,t); axis equal;
x=p(1,:)';
y=p(2,:)';
% начальное смещение точек мембраны в форме пирамиды
u0=(2-abs(x)-abs(y)-abs(abs(y)-abs(x)))/2;
ut0= 0;                 % начальная скорость точек мембраны 0
n=31; tlist=linspace(0,3,n);  % вектор моментов времени
uw=hyperbolic(u0,ut0,tlist,bw,p,e,t,1,0,0,1);   % решение уравнения
% анимация решения
umax=max(max(uw));      % максимальный элемент матрицы смещений
umin=min(min(uw));      % минимальный элемент матрицы смещений
newplot                 % создаем новое графическое окно
clear M;
for i=1:n,
    pdesurf(p,t,uw(:,i));           % рисуем кадр – график поверхности
    axis equal;
    axis([-1 1 -1 1 umin umax]);    % приводим кадр к общему масштабу
    M(i)=getframe;          % запоминаем кадр в массив
end
movie(M,4);                 % проигрываем массив кадров 4 раза
```

На следующем рисунке показана форма мембраны в различные моменты времени

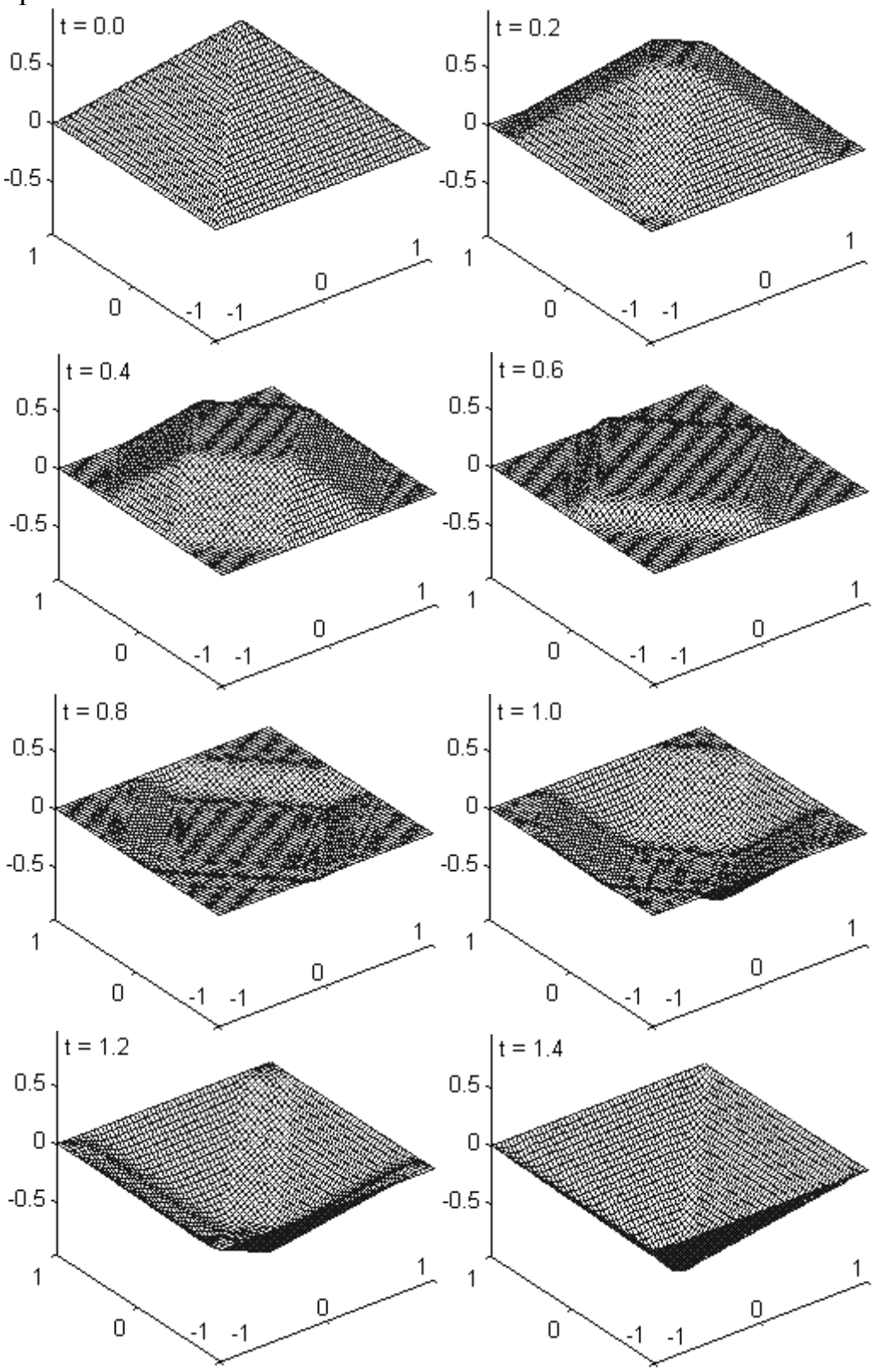


Рис. 20

Для построения этих графиков мы создали сценарий `drawFrame1.m`, в котором переменная `myframe` содержит номер отображаемого кадра.

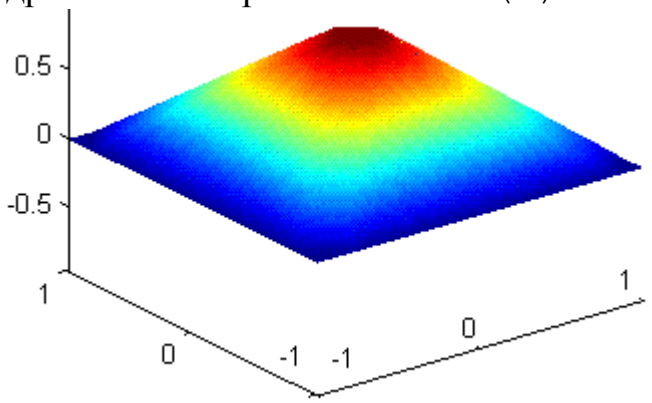
```
% сценарий рисование кадра с номером myframe
myframe=15;
pdeplot(p,[],t,'xydata',uw(:,myframe),'xystyle','off', ...
```

```
     'zdata',uw(:,myframe),'zstyle','continuous','colorbar','off',
     'mesh','on','xygrid','on');
axis equal;     % равенство длин засечек на осях
                % задаем перед заданием пределов осей
axis([-1 1 -1 1 umin umax]);
colormap bone;
s=sprintf('t = %2.1f',tlist(myframe));
text(-1,0.9,0.9,s);       % вывод момента времени кадра
```

Функция **pdeplot(…)** является общей функцией, предназначенной для построения графиков решения, полученного с помощью функций PDE TOOLBOX. Функция **pdesurf(…)**, используемая нами ранее, является ее частным случаем. Используя ее, кадры анимации можно строить следующим образом

colormap 'default';
pdesurf(p,t,uw(:,2));
axis([-1 1 -1 1 umin umax]);

Здесь 2 – номер кадра в момент времени `tlist(2)=0.1`.



Опции функции **pdeplot(…)**, использованные нами в сценарии `drawFrame1.m`, предназначены для построения черно-белой каркасной поверхности.

Общие положения решения ДУЧП с использованием функций PDE Toolbox

Здесь мы более подробно разберем шаги решения краевых задач с использованием функций пакета, описанные выше.

Задание геометрии области.

Имеются два способа определения геометрии области. Первый заключается в параметрическом задании частей границ области в файл–функции, имеющей определенный формат. Второй, использованный нами в предыдущих примерах, состоит в задании области в среде pdetool, последующем экспорте информации в переменные рабочей среды и преобразовании в формат, понятный другим функциям пакета.

Нарисуем в среде pdetool прямоугольник с центром в начале координат шириной два и высотой один. Выберем меню `Draw – Export Geometry Description, Set Formula, Labels…` Появится диалоговое окно `Export`, изображенное на следующем рисунке.

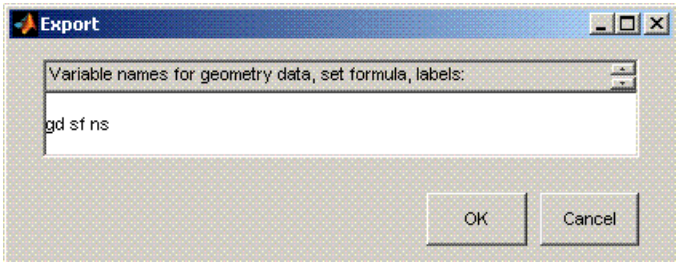


Рис. 21

Оно предлагает сохранить модель (конструктивную блочную геометрию – CSG модель) в глобальных переменных рабочей среды:

- матрицу геометрии области (Geometry Description matrix) в массиве **gd**;
- формулу связи геометрических примитивов в строке **sf**;
- соответствие между столбцами **gd** и названиями областей в **sf** в массиве **ns**.

Нажмите ОК и займитесь изучением содержимого экспортированных массивов.

Число столбцов матрицы геометрии области **gd** совпадает с числом геометрических примитивов, составляющих область. В рассматриваемом примере матрица состоит из одного столбца, соответствующего прямоугольнику.

```
>> gd
gd =
     3.0000        % тип примитива (3 = прямоугольник)
     4.0000        % число сторон многоугольника
     1.0000        % x координата 1-й вершины (правой нижней)
    -1.0000        % x координата 2-й вершины (левой нижней)
    -1.0000        % x координата 3-й вершины (левой верхней)
     1.0000        % x координата 4-й вершины (правой верхней)
    -0.5000        % y координата 1-й вершины (правой нижней)
    -0.5000        % y координата 2-й вершины (левой нижней)
     0.5000        % y координата 3-й вершины (левой верхней)
     0.5000        % y координата 4-й вершины (правой верхней)
```

Первый элемент столбца определяет тип геометрического примитива (всего допустимы четыре типа геометрических примитивов: круг, многоугольник, прямоугольник и эллипс). Первый элемент, равный трем, означает, что данный столбец соответствует прямоугольнику. Во второй строке указано количество сторон многоугольника (прямоугольник является частным случаем многоугольника, который имеет специальный формат хранения геометрии). В остальные элементы столбца записаны координаты вершин прямоугольника – вначале *x* координаты, начиная с правой нижней с обходом по часовой стрелке, потом *y* координаты.

Структура столбцов для разных примитивов приведена в следующей таблице.

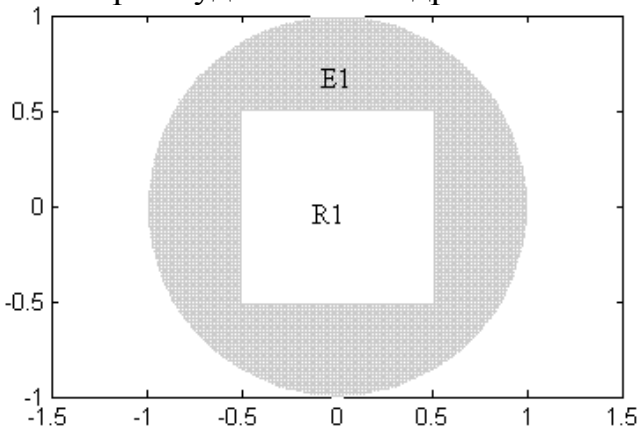
	Круг	**Многоугольник**	**Прямоугольник**	**Эллипс**
Номер типа	1	2	3	4
2-я строка	$X_{центра}$	Число сторон	Число сторон	$X_{центра}$
3-я строка	$Y_{центра}$	X_1	X_1	$Y_{центра}$
4-я строка	$R_{круга}$	Y_1	…	$X_{полуось}$
…		…	X_4	$Y_{полуось}$

		…	Y_1	угол поворота
		…	…	
		Y_n	Y_4	

Координаты вершин многоугольника записываются в столбец в порядке их создания. Другие CSG элементы имеют вид

```
>> sf                % формула области (имя единственной подобласти R1)
sf =
R1
>> ns
ns =
    82              % char(82)=R (код первого символа имени R)
    49              % char(49)=1 (код второго символа имени 1)
```

Для пояснения построим еще одну область – единичный круг с центром в начале координат, из которого удаляется квадрат меньшего размера.



Экспортируем, как описано выше, модель в переменные рабочей среды `gd, sf, ns` и изучим полученные матрицы.

```
>>gd
gd =
    4.0000    3.0000
         0    4.0000
         0    0.5000
    1.0000   -0.5000
    1.0000   -0.5000
         0    0.5000
         0   -0.5000
         0   -0.5000
         0    0.5000
         0    0.5000
```

Область сконструирована из двух примитивов и матрица `gd` состоит из двух столбцов. Первый соответствует эллипсу (номер типа 4), второй – прямоугольнику (номер типа 3). Номера типов записаны в первой строке каждого столбца.

В строковой переменной `sf` записана формула конструирования области – из области круга (эллипса) вычитается область квадрата.

```
>> sf
sf =
E1-R1
```

В массиве `ns` хранится соответствие между столбцами матрицы `gd` и названиями областей

```
>> ns
ns =
```

```
    69    82
    49    49
```

Столбцы матрицы `ns` хранят коды символов имен примитивов, из которых сконструирована область – первый столбец `E1`, второй – `R1`.

char(ns)

```
ans =
ER
11
```

Если переименовать области и эллипсу дать имя `Ellipse`, а квадрату – `Rect`, то после экспорта конструктивной блочной геометрии массив `gd` останется без изменений, строковая переменная `sf` будет равна

>> sf

```
sf =
Ellipse-Rect
```

Массив `ns` будет хранить коды символов имен примитивов

>> ns

```
ns =
    69    82
   108   101
   108    99
   105   116
   112    32
   115    32
   101    32
```

>> char(ns')

```
ans =
Ellipse
Rect
```

При этом, второй столбец, соответствующий имени `Rect`, дополнен пробелами (код символа 32) для выравнивания длин столбцов матрицы `ns`.

Функции, связанные с триангуляцией и заданием граничных условий, используют другой формат описания геометрии области – матрицу декомпозиционной геометрии (Decomposed Geometry matrix). Она хранит информацию о частях границы области, которые могут быть отрезком, частью дуги окружности или эллипса. Переход к декомпозиционной матрице производится при помощи функции **decsg**, входными аргументами которой являются переменные конструктивной блочной геометрии, в нашем случае `gd`, `sf` и `ns`, а выходным – матрица декомпозиционной геометрии. Каждый столбец этой матрицы соответствует элементу границы. В примерах 1 и 2 матрицу декомпозиционной геометрии мы получали одновременно с экспортом матрицы граничных условий.

Преобразуйте экспортированные величины (рассматриваемая область – прямоугольник, построенный выше) `gd`, `sf` и `ns`, вызвав функцию **decsg** из командной строки:

>> dl=decsg(gd,sf,ns)

```
dl =
    2.0000    2.0000    2.0000    2.0000
    1.0000   -1.0000   -1.0000    1.0000
   -1.0000   -1.0000    1.0000    1.0000
   -0.5000   -0.5000    0.5000    0.5000
   -0.5000    0.5000    0.5000   -0.5000
```

```
     0         0         0         0
1.0000    1.0000    1.0000    1.0000
```

Прямоугольник представляется четырьмя отрезками. Двойка в первом элементе столбца означает, что столбец соответствует отрезку, второй и третий элементы содержат абсциссы начала и конца отрезка, а четвертый и пятый — ординаты. Преобразование в декомпозиционную геометрию предполагает, что вся область разбита на некоторые непересекающиеся подобласти, имеющие собственные идентификаторы (номера). Идентификаторы подобластей, расположенных слева и справа от элемента границы (отрезка или дуги), записаны в шестом и седьмом элементе. В нашем примере есть только одна область – прямоугольник (идентификатор единица). Слева от отрезков, представляющих границу прямоугольника, областей нет – идентификатор ноль.

В случае, когда столбец соответствует дуге эллипса или окружности, элементы столбца будут содержать информацию о центре и радиусе окружности или величине полуосей эллипса и угле поворота.

С помощью функции **pdegplot** можно построить контур области (границу) в отдельном графическом окне. Входным аргументом функции является матрица декомпозиционной геометрии (в нашем случае `dl`)

>>pdegplot(dl); axis equal

Функциям PDE TOOLBOX вместо матрицы декомпозиционной геометрии можно передавать имя файл – функции, вычисляющей координаты точек участков границы области по их параметрическим уравнениям. Из матрицы декомпозиционной геометрии такую функцию можно сгенерировать автоматически при помощи функции **wgeom**. Ее входными аргументом является матрица декомпозиционной геометрии и имя М-файла, в который следует записать файл-функцию. Если файл-функцию создать не удалось, то **wgeom** вернет минус единицу.

Создадим файл-функцию **recgeom**, которая описывает прямоугольную область, задаваемую матрицей `dl`

>>wgeom(dl, 'recgeom')

```
ans =
     3
```

В текущем каталоге MATLAB появится файл `recgeom.m`, структура которого приведена ниже

```
function [x,y]=recgeom(bs,s)
%RECGEOM    Gives geometry data for the recgeom PDE model.
%
%   NE=RECGEOM возвращает количество граничных сегментов
%
%   D=RECGEOM(BS) для граничного сегмента с номером BS возвращает матрицу
%   из одного столбца каждый элемент которого является
%   Row 1 contains the start parameter value.
%   Row 2 contains the end parameter value.
%   Row 3 contains the number of the left-hand regions.
%   Row 4 contains the number of the right-hand regions.
%
%   [X,Y]=RECGEOM(BS,S) gives coordinates of boundary points.
```

```
%   BS specifies the boundary segments and
%   S the corresponding parameter values.
%   BS may be a scalar.

nbs=4;

if nargin==0
  x=nbs; % number of boundary segments
  return
end

d=[
  0 0 0 0 % start parameter value
  1 1 1 1 % end parameter value
  0 0 0 0 % left hand region
  1 1 1 1 % right hand region
];

bs1=bs(:)';       % вектор строка или столбец bs становится строкой bs1

if find(bs1<1 | bs1>nbs)   % двойной знак || используется для скаляров
  error('Non-existent boundary segment number')
end

if nargin==1
  x=d(:,bs1);
  return
end

x=zeros(size(s));
y=zeros(size(s));
[m,n]=size(bs);
if m==1 & n==1,
  bs=bs*ones(size(s)); % expand bs
elseif m~=size(s,1) || n~=size(s,2),
  error('bs must be scalar or of same size as s');
end

if ~isempty(s),

% boundary segment 1
ii=find(bs==1);
if length(ii)
x(ii)=(-1-(1))*(s(ii)-d(1,1))/(d(2,1)-d(1,1))+(1);
y(ii)=(-0.5-(-0.5))*(s(ii)-d(1,1))/(d(2,1)-d(1,1))+(-0.5);
end

% boundary segment 2
ii=find(bs==2);
if length(ii)
x(ii)=(-1-(-1))*(s(ii)-d(1,2))/(d(2,2)-d(1,2))+(-1);
y(ii)=(0.5-(-0.5))*(s(ii)-d(1,2))/(d(2,2)-d(1,2))+(-0.5);
end

% boundary segment 3
ii=find(bs==3);
if length(ii)
x(ii)=(1-(-1))*(s(ii)-d(1,3))/(d(2,3)-d(1,3))+(-1);
y(ii)=(0.5-(0.5))*(s(ii)-d(1,3))/(d(2,3)-d(1,3))+(0.5);
end

% boundary segment 4
ii=find(bs==4);
```

```
if length(ii)
x(ii)=(1-(1))*(s(ii)-d(1,4))/(d(2,4)-d(1,4))+(1);
y(ii)=(-0.5-(0.5))*(s(ii)-d(1,4))/(d(2,4)-d(1,4))+(0.5);
end

end
```

Основное назначение файл - функции **[x,y]=recgeom(bs,s)** состоит в генерировании координат (`x,y`) точки участка границы с номером `bs` и параметром `s`. Фактически в коде этой функции записаны параметрические уравнения каждого участка границы. Чтобы вычислить координаты точки надо задать номер участка границы и значение ее параметра в параметрическом уравнении кривой.

Вызов **recgeom** без аргументов возвращает количество граничных отрезков. Вызов **recgeom(номер_участка_границы)** возвращает столбец, элементы которого являются начальным значением параметра в параметрическом уравнении участка границы, конечным значением параметра и номерами подобластей, расположенных слева и справа от указанного участка границы.

Локальная переменная `nbs` содержит число частей границы. Локальная матрица `d` содержит информацию о параметризации – число столбцов `d` равно числу участков границы, первый и второй элементы каждого столбца содержат начальное и конечное значения параметра (ноль и единица), в третьем и четвертом элементе записаны номера примитивов, расположенных по левую и правую сторону от участка границы (по направлению возрастания параметра). В блоках кода с комментарием `% boundary segment …` происходит поиск нужной части границы и вычисление координат (`x, y`) точки. Длины массивов `x, y` совпадают с длиной входного массива значений параметров `s`.

Область должна задаваться как минимум двумя граничными кривыми! Например, использование только одной пары уравнений $x = \cos t$, $y = \sin t$ для задания границы круга в теле функции декомпозиционной геометрии приведет к ошибке.

Умение создавать файл - функции с декомпозиционной геометрией области необходимо в том случае, когда область не может быть сконструирована только с использованием стандартных геометрических примитивов, таких как круг, эллипс, квадрат, прямоугольник или многоугольник. В своих файл – функциях вы можете использовать любые выражения и функции MatLab для вычисления координат точек границы.

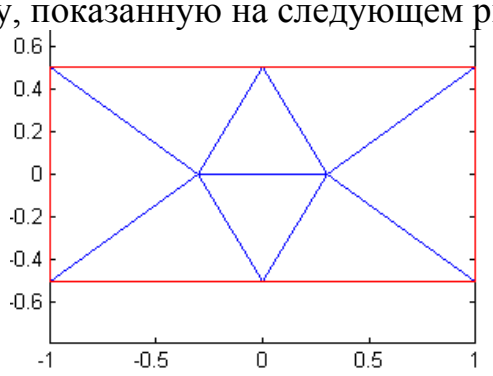
Триангуляция

Функция **initmesh** разбивает плоскую область на треугольники. Ее входным аргументом является матрица декомпозиционной геометрии. Команда **[p, e, t] = initmesh(dl)** приводит к заполнению трех матриц **p, e, t**, которые содержат информацию о триангуляции. Функция **initmesh** может содержать дополнительные аргументы, задаваемые парами: название

свойства, значение. Например, свойство `Hmax` устанавливает максимально допустимое значение длины стороны треугольного элемента. Полученную треугольную сетку можно отобразить в отдельном окне при помощи **pdemesh**, входными аргументами которой являются вышеперечисленные массивы. Например, последовательность команд

```
>>[p,e,t]=initmesh(dl,'Hmax',1.0);
>> pdemesh(p,e,t); axis equal
```

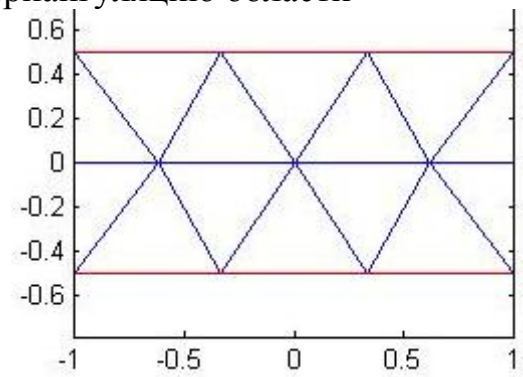
создает и выводит сетку, показанную на следующем рисунке



Вместо матрицы декомпозиционной геометрии аргументом функции **initmesh** может быть имя файл - функции с декомпозиционной геометрией области. Например, команды

```
[p,e,t]=initmesh('recgeom','Hmax',0.7);
pdemesh(p,e,t); axis equal
```

создают следующую триангуляцию области



Обычно значение параметра `Hmax` равно 1/10 некоторого характерного размера области (ширины, высоты, диаметра и т.д.).

Чтобы понять структуру массивов **p, e, t** перейдем в среду pdetool, нарисуем прямоугольник 2×1, из меню `Mesh - Parameters...`, вызовем диалоговое окно `Mesh Parameters` и в нем установим достаточно крупный размер стороны треугольного элемента (см. следующий рисунок)

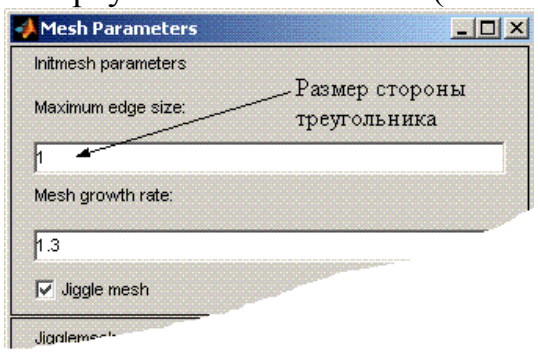


Рис. 22

Затем включим режимы нумерации узлов с помощью меню `Mesh - Show Node Labels` и нумерации элементов с помощью меню `Mesh - Show Triangle Labels` (рис. 23)

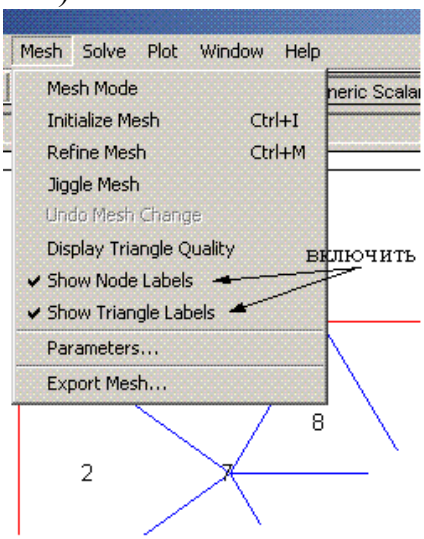


Рис. 23

Включим режим сетки (меню `Mesh - Mesh Mode`). В результате получим следующее изображение сетки (рис. 24)

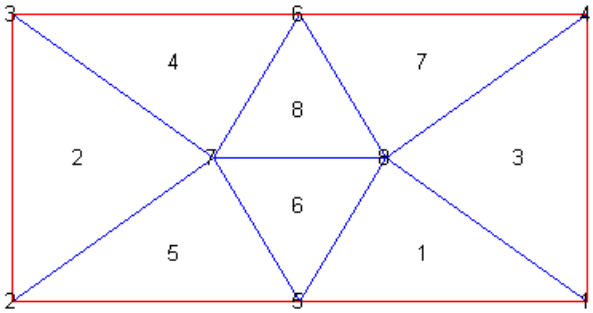


Рис. 24

Далее экспортируем сетку в глобальные массивы **p, e, t** при помощи пункта меню `Export Mesh...`,

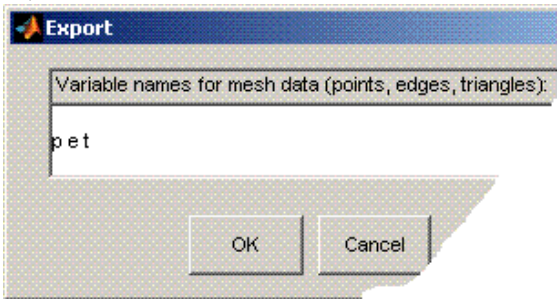


Рис. 25

Исследуем содержимое этих массивов в командном окне. В матрице **p** первая и вторая строки содержат *x* и *y* координаты узлов сетки

>> p

```
p =
    1.0000   -1.0000   -1.0000    1.0000         0         0   -0.3000    0.3000
   -0.5000   -0.5000    0.5000    0.5000   -0.5000    0.5000         0         0
```

Номер столбца матрицы **p** отвечает номеру узла сетки (номера узлов вы видите на рисунке 24).

Матрица **e** содержит информацию об одномерных элементах триангуляции, расположенных на границе.

>> e

```
e =
    1.0000    5.0000    2.0000    3.0000    6.0000    4.0000
    5.0000    2.0000    3.0000    6.0000    4.0000    1.0000
         0    0.5000         0         0    0.5000         0
    0.5000    1.0000    1.0000    0.5000    1.0000    1.0000
    1.0000    1.0000    2.0000    3.0000    3.0000    4.0000
         0         0         0         0         0         0
    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000
```

Первая и вторая строки содержат номера начальной и конечной точек элементов (отрезков или дуг), принадлежащих границе области. Третья и четвертая строки содержат начальное и конечное значение параметра в параметрическом уравнении участка границы. Пятая строка содержит номер участка границы, которому принадлежит граничный элемент (номер столбца в матрице декомпозиционной геометрии `dl`). Шестая и седьмая строки содержат номера подобластей расположенных слева и справа от элемента.

В матрице **t** хранится информация о треугольных элементах разбиения области. Номер столбца матрицы t соответствует номеру треугольного элемента на рисунке 24.

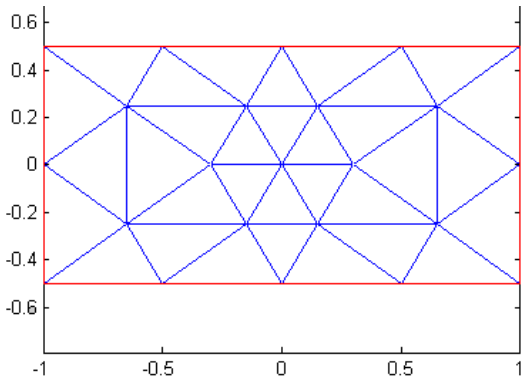
>> t

```
t =
     5     3     1     6     2     7     4     6
     1     2     4     3     5     5     6     7
     8     7     8     7     7     8     8     8
     1     1     1     1     1     1     1     1
```

Первые три элемента столбца содержат номера узлов треугольного элемента в направлении обхода против часовой стрелки, а четвертый – номер подобласти, которой принадлежит треугольный элемент.

Для уменьшения шага сетки используется функция **refinemesh**. Первым входным аргументом является матрица или функция декомпозиционной геометрии, далее передаются массивы с информацией о триангуляции. В выходных аргументах возвращаются массивы, соответствующие измельченной сетке. По умолчанию сторона каждого элемента делится пополам, каждый треугольный элемент исходной сетки разбивается на четыре части. Команды, приведенные ниже, производят вышеописанное уменьшение шага сетки

```
[p1,e1,t1]=refinemesh(dl,p,e,t);
pdemesh(p1,e1,t1); axis equal
```



Функция **refinemesh** позволяет задать ряд дополнительных параметров, управляющих процедурой дробления сетки, в частности, делить только некоторые треугольные элементы или подобласти. Кроме того, можно выбрать другой алгоритм уменьшения шага сетки, который делит на две равные части наибольшую сторону конечного элемента. Первым входным аргументом может быть не только матрица декомпозиционной геометрии, но и имя файл - функции, описывающей декомпозиционную геометрию области.

Граничные условия и коэффициенты уравнения

Граничные условия в PDE Toolbox задаются двумя способами, либо при помощи матрицы граничных условий (Boundary Condition matrix), либо в файл - функции. Число столбцов матрицы граничных условий совпадает с числом столбцов в матрице декомпозиционной геометрии, т. е. каждый столбец отвечает участку границы области. На каждой части границы может быть задано условие одного из типов: Дирихле `h*u = r`, или условие Неймана $\langle \mathbf{n}, (c \cdot \nabla u) \rangle + q u = g$. Здесь (если `u` – скалярная функция, т.е. решается одно уравнение) `h, r, q, g` – постоянные или функции переменных *x,y*, параметр `c` может быть постоянной или матрицей `2 x 2`, **n** – вектор нормали к границе, $\langle\ \rangle$ - скалярное произведение векторов.

Разберем структуру матрицы граничных условий. Настроим среду pdetool на решение скалярного (см. рис. 26) эллиптического уравнения, установив переключатель `Elliptic` в окне `PDE Specification` (см. рис. 14).

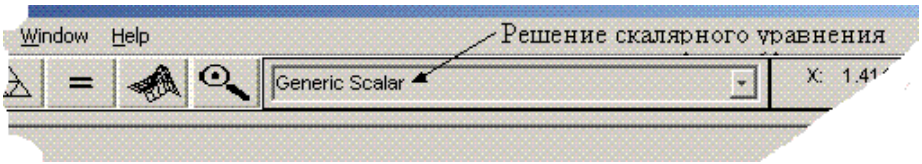


Рис. 26

Зададим на всех участках границы прямоугольника условие Дирихле, например x^2+y^2. Для этого в диалоговом окне `Boundary Condition` введем 1 в поле `h` и *x.^2 + y.^2* в поле `r`. Экспортируем в рабочую среду матрицу граничных условий, выбрав в меню `Boundary – Export Decomposed Geometry, Boundary Cond's...` Появляется диалоговое окно `Export`, в котором по умолчанию стоят имена `g` и `b` – первое для

массива декомпозиционной геометрии и второе – для хранения матрицы граничных условий. Матрицу декомпозиционной геометрии мы уже ранее строили с помощью функции **decsg**. Нажмем ОК и в рабочей среде появился двумерный массив b.

b

```
b =
     1     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1
     9     9     9     9
    48    48    48    48
    48    48    48    48
    49    49    49    49
   120   120   120   120
    46    46    46    46
    94    94    94    94
    50    50    50    50
    43    43    43    43
   121   121   121   121
    46    46    46    46
    94    94    94    94
    50    50    50    50
```

Структура матрицы граничных условий приспособлена для хранения граничных условий в задачах, описываемых системой дифференциальных уравнений. Каждый столбец соответствует участку границы. В нашем случае область является прямоугольником с одинаковыми граничными условиями Дирихле, и матрица состоит из четырех столбцов.

Столбец условно делится на две части, верхняя часть столбца матрицы граничных условий содержит информацию о типе граничных условий и длинах формул. В нижней части записаны коэффициенты и формулы граничных условий. При этом символы строки с формулой хранятся в последовательно идущих элементах в виде своих кодов. В случае уравнения (а не системы) и граничного условия Дирихле формулы хранятся в элементах столбца, начиная с седьмого, в чем несложно убедиться, преобразовав элементы нижней части матрицы из кодов в символы

char(b(7:end,:)')

```
ans =
001x.^2+y.^2
001x.^2+y.^2
001x.^2+y.^2
001x.^2+y.^2
```

Здесь для наглядности представления формул мы транспонировали матрицу граничных условий. Первые два элемента зарезервированы под коэффициенты условия Неймана (они равны нулю, поскольку поставлено условие Дирихле). Третий элемент содержит значение `h` (равное единице), а остальные являются символами выражения `x.^2+y.^2`.

Матрица граничных условий имеет столько же столбцов, сколько матрица декомпозиционной геометрии – т.е. столько, сколько участков границы. Вид матрицы отражает структуру граничных условий – для каждого участка свое граничное условие и, следовательно, свой столбец со

своей структурой. Для скалярного уравнения на участке границы можно поставить граничное условие Неймана или условие Дирихле. Условие Неймана имеет вид $\frac{\partial u}{\partial \mathbf{n}} + q\,u = g$, а Дирихле – вид $h \cdot u = r$. И поэтому столбец матрицы должен содержать значения коэффициентов `q, g, h` и `r`.

При решении одного скалярного уравнения каждый столбец матрицы граничных условий будет содержать следующие элементы:

№ строки	Элемент столбца
1	Размерность системы уравнений = 1
2	Кол-во условий Дирихле (0 или 1)
3	n_q – к–во символов представляющих `q`
4	n_g – к –во символов представляющих `g`
5	n_h – к –во символов представляющих `h` (если условия Дирихле есть)
6	n_r – к –во символов представляющих `r` (если условия Дирихле есть)
…	Текст формулы `q` (коды символов)
…	Текст формулы `g` (коды символов)
…	Текст формулы `h` (коды символов) (если условия Дирихле есть)
…	Текст формулы `r` (коды символов) (если условия Дирихле есть)
…	Нули, если надо выровнять длину столбцов матрицы.

Например, для одного ДУЧП и граничного условия $\frac{\partial u}{\partial \mathbf{n}} = -x^2$ этот вектор – столбец будет иметь вид $\left[1\ 0\ 1\ 5\ '0'\ '-x.\hat{}2'\right]$. Для граничного условия Дирихле $u\big|_{\partial\Omega} = x^2 - y^2$ вектор – столбец матрицы граничных условий, соответствующий участку границы на котором задано это условие, будет равен $\left[1\ 1\ 1\ 1\ 1\ 9\ '0'\ '0'\ '1'\ 'x.\hat{}2 - y.\hat{}2'\right]$.

Символьные массивы (строки в одинарных кавычках) автоматически заменяются кодами символов. Однако лучше сразу заменить символьные массивы числовыми массивами, содержащими коды символов. Например, строку

s='x.^2-y.^2'

можно сразу заменить числовым вектором

c=double(s)

```
c =
   120    46    94    50    45   121    46    94    50
```

Для системы уравнений матрица граничных условий имеет более сложный вид, с которым вы можете познакомиться на странице справки функции **assemb**.

При вызове функций пакета PDE TOOLBOX вместо матрицы граничных условий можно использовать функцию граничных условий. Ее можно сгенерировать автоматически из матрицы граничных условий с помощью функции **wbound**. Ее аргументами является матрица граничных условий и имя генерируемой файл – функции граничных условий. Выполним команду

wbound(b,'myBound')

Функция **wbound** вернет -1, если файл граничных условий создать не удалось.

```
function [q,g,h,r]=myBound(p,e,u,time)
%MYBOUND   Boundary condition data.
%
bl=[
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
9 9 9 9
48 48 48 48
48 48 48 48
49 49 49 49
120 120 120 120
46 46 46 46
94 94 94 94
50 50 50 50
43 43 43 43
121 121 121 121
46 46 46 46
94 94 94 94
50 50 50 50
];
if any(size(u))
  [q,g,h,r]=pdeexpd(p, e, u, time, bl);
else
  [q,g,h,r]=pdeexpd(p, e, time, bl);
end
```

Как видим, созданная функция содержит локальный массив `bl` полностью совпадающий с матрицей граничных условий. Переменная `bl` затем передается недокументированной функции **pdeexpd(…)**. Заметим, что функция **any(A)**, использованная в теле функции граничных условий возвращает логическую единицу (`true`), если в векторе есть хотя бы один ненулевой элемент.

Хотя описания функции **pdeexpd(…)** в справочной системе нет, вы можете посмотреть ее код, выполнив команду

type pdeexpd

Если вы вручную захотите создать свою функцию граничных условий, то в шаблон функции **myBound**, приведенный выше, вы должны будете вставить свою матрицу граничных условий. Заметим однако, что создание функции граничных условий в большинстве случаев не требуется и можно ограничится созданием матрицы граничных условий.

Солверы.

Приближенным решением граничной задачи является вектор значений в узлах сетки. Его создают специальные функции пакета PDE Toolbox называемые солверами (solver). Перед ее вызовом должна быть создана матрица или файл–функция граничных условий `b`, заданы триангуляция в массивах `p, e, t` и заданы коэффициенты уравнения (разумеется, переменные могут называться по-другому). Если решается нестационарная задача, то необходимо создать функции, определяющие решение в начальный момент времени и вектор со значениями времени, в которые следует найти решение. Затем эти данные передаются функциям солверам. Они находят приближенное решение задачи в узлах сетки. Солверы реализованы в специальных файл - функциях – для каждого типа задачи свой солвер. В следующей таблице перечислены основные солверы

Имя функции	Описание типа решаемых задач
assempde	Эллиптические уравнения и системы
parabolic	Параболические уравнения и системы
hyperbolic	Гиперболические уравнения и системы
pdenonlin	Нелинейные стационарные уравнения
adaptmesh	Адаптивная генерация сетки и решение эллиптических уравнений и систем с заданной точностью
pdeeig	решение эллиптических задач на собственные значения

Интерфейс солвера **assempde** является наиболее универсальным. Она решает уравнение (или систему) вида $-div(c \cdot grad(u)) + a \cdot u = f$. Первым его входным аргументом является матрица или строка с именем файл – функции граничных условий. Далее указываются матрицы с информацией о триангуляции и коэффициенты уравнения `c, a, f`. Ее вызовы с одним выходным аргументом

```
u = assempde(b, p, e, t, c, a, f);              % b - матрица
u = assempde('bouncond', p, e, t, c, a, f);     % bouncond - функция
```

приводят к записи в вектор `u` значений решения в узлах сетки, координаты которых хранятся в `p`.

Входные аргументы солверов **parabolic** и **hyperbolic** такие же, как у **assempde**. Кроме того, задается вектор значений решения u_0 в начальный момент времени (в случае гиперболического уравнения еще и вектор u_{t0} с начальным значением производной решения по времени), вектор `tlist` моментов времени, в которые требуется найти решение, и коэффициент `d` уравнения при второй производной по времени. Эти солверы возвращают матрицу `u` с решением, каждый столбец которой есть вектор со значениями приближенного решения в узлах сетки в соответствующие моменты времени, указанные в `tlist`:

```
u = parabolic(u0, tlist, b, p, e, t, c, a, f, d)
```

u = hiperbolic(u0, ut0, tlist, b, p, e, t, c, a, f, d)

Об использовании других солверов вы можете узнать из справочной системы MatLab. Один из них **pdenonlin** мы неявно использовали, когда решали нелинейное уравнение при построении минимальной поверхности. В том примере, перед тем как решать уравнение, мы выбрали меню `Solve – Parameters` и в открывшемся окне устанавливали флажок `Use nonlinear solver` (рис.15).

Визуализация результатов

Функции **pdegplot** и p**demesh** предназначены для графического отображения геометрии области и сетки, их использование описано выше. Основной функцией для визуализации решения является **pdeplot**, которая позволяет управлять видом получаемых графиков при помощи ряда дополнительных параметров. Функции **pdecont** и **pdesurf**, строящие линии уровня решения или трехмерный график, реализуют частные случаи обращения к **pdeplot**. Входными аргументами **pdecont** и **pdesurf** являются матрица `p` с координатами узлов, матрица `t` соответствия глобальной и локальной нумераций и вектор `u` со значениями решения в узлах. Четвертым дополнительным аргументом **pdecont** может являться вектор значений, которые требуется отобразить линиями уровня, или их число.

Первые три аргумента функции **pdeplot** являются массивами `p`, `e` и `t` с информацией о триангуляции. Затем парами задаются имя параметра (свойства) и его значение. В следующей таблице описаны некоторые из этих параметров

Имя параметра	Значение
`xydata`	Вектор со значениями решения в узлах для построения двумерного графика
`xystyle`	`off, flat, interp` (по умолчанию) – способы заливки контуров
`contour`	`on, off` (по умолчанию) – отображение или скрытие контурных линий
`zdata`	Вектор со значениями решения в узлах для построения трехмерного графика
`colormap`	`cool, hot, gray, bone`, ... – цветовые палитры
`mesh`	`on, off` (по умолчанию) – отображение конечно элементной сетки
`colorbar`	`off, on` (по умолчанию) – вывод шкалы соответствия цвета и значения
`title`	Строка с заголовком
`levels`	Число линий уровня или вектор со значениями решения, отображаемого линиями уровня

Использование одной из пар ‘`xydata`’ или ‘`zdata`’ обязательно, поскольку содержит вектор значений решения в узлах сетки.

Для иллюстрации использования описанных выше функций решим несколько задач в полукруге. Центр полукруга находится в начале координат, радиус единица, выбрана верхняя часть круга. Решим в этой области уравнение Пуассона, Лапласа, задачу о колебании полукруглой мембраны и найдем форму минимальной поверхности. Вначале создадим функцию декомпозиционной геометрии области, при этом учтем, что область должна задаваться как минимум двумя граничными кривыми.

Функция декомпозиционной геометрии полукруга

```
function [x,y]=halfcirclegeom(bs,s)
% декомпозиционная функция полукруга

nbs=2;
if nargin==0,
  x=nbs; % number of boundary segments
  return
end

d=[
  0 0   % start parameter value
  1 1   % end parameter value
  1 1   % left hand region
  0 0   % right hand region
];

bs1=bs(:)';

if find(bs1<1 | bs1>nbs),
  error('Non-existent boundary segment number')
end

if nargin==1,
  x=d(:,bs1);
  return
end

x=zeros(size(s));
y=zeros(size(s));
[m,n]=size(bs);
if m==1 & n==1,
  bs=bs*ones(size(s)); % expand bs
elseif m~=size(s,1) | n~=size(s,2),
  error('bs must be scalar or of same size as s');
end

if ~isempty(s),
      % boundary segment 1
      ii=find(bs==1);
      if length(ii)
            x(ii)=cos(pi*s(ii));
            y(ii)=sin(pi*s(ii));
      end

      % boundary segment 2
      ii=find(bs==2);
      if length(ii)
            x(ii)=-1+2*s(ii);
            y(ii)=0;
      end
end
```

```
% ----------------- конец тела функции ----------------
```

Проверяем, что граница области создана правильно (следующий левый рисунок)

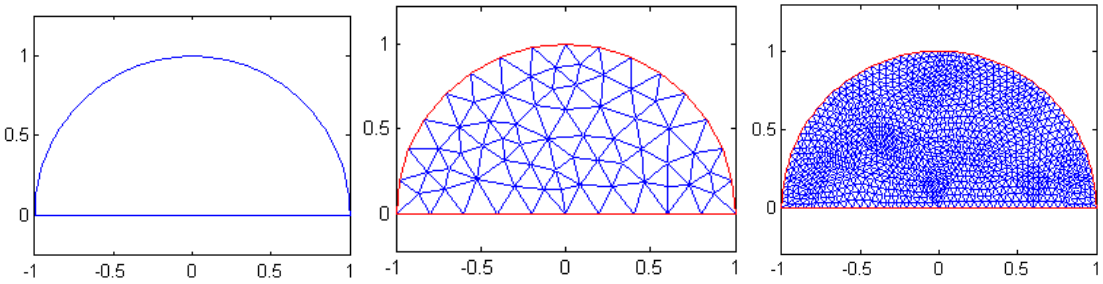
pdegplot('halfcirclegeom'), axis equal

Строим и рисуем сетку (средний рисунок)

[p, e, t] = initmesh('halfcirclegeom');
pdemesh(p,e,t); axis equal;

Сгустим и нарисуем сетку (правый рисунок)

[p,e,t]=refinemesh('halfcirclegeom',p,e,t);
[p,e,t]=refinemesh('halfcirclegeom',p,e,t);
pdemesh(p,e,t); axis equal;



Функцию **halfcirclegeom** и вектора **[p, e, t]** будем использовать для решения краевых задач в полукруге, приводимых в следующих примерах.

Пример 1. Решим уравнение Пуассона $\Delta u = -1$ с нулевыми граничными условиями $u\big|_{\partial\Omega} = 0$.

Для граничных условий Дирихле $h \cdot u = r$ на обеих участках границы (на диаметре и полуокружности) имеем `h=1, r=0`. Матрицу граничных условий можно построить командой

b=[1 1; 1 1; 1 1; 1 1; 1 1; 1 1; 48 48; 48 48; 49 49; 48 48]

```
b =
     1     1
     1     1
     1     1
     1     1
     1     1
     1     1
    48    48
    48    48
    49    49
    48    48
```

Здесь числа 48 и 49 являются кодами символов 0 и 1, представляющих выражения для `r` и `h`. Проверим

char(b(7:end,:)')

```
ans =
0010
0010
```

Эллиптическое уравнение $-div(c \cdot grad(u)) + a \cdot u = f$ принимает требуемую форму при `a=0, c=1, f=1`. Задаем коэффициенты и правую часть уравнения $-\Delta u = f$

a=0; `% коэффициент уравнения`

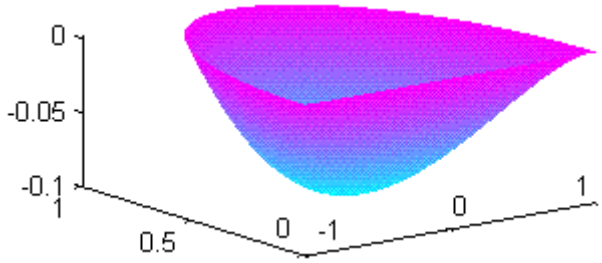
```
c=1;          % коэффициент уравнения
f=1;          % Строка с формулой правой части уравнени
```

Для решения эллиптических уравнений используется функция **assempde**. Ей следует передать аргументы – матрицу граничных условий `b`, массивы `p, e, t,` содержащие информацию о триангуляции, и коэффициенты уравнения `c, a, f`.

```
u=assempde(b,p,e,t,c,a,f);
```

Строим график поверхности решения `u(x,y)`

```
pdesurf(p,t,u);
```



Пример 2. Решим уравнение Лапласа $\Delta u = 0$ в полукруге с граничным условием $u\big|_{\partial\Omega} = |x|$.

Посмотрим какие коды имеют символы граничного условия $h \cdot u = r$, где `r=abs(x)`. Построим строку s, содержащей формулу, и преобразуем ее в числовой массив

```
s='abs(x)'
s =
abs(x)
double(s)
ans =
    97    98   115    40   120    41
```

Теперь можно создать матрицу граничных условий. На шестом месте столбцов матрицы граничных условий хранятся длины формул условия Дирихле. Количество символов в формуле `abs(x)` равно шести. Поэтому

```
b=[1 1; 1 1; 1 1; 1 1; 1 1; 6 6; 48 48; 48 48; 49 49;...
                97 97; 98 98; 115 115; 40 40; 120 120; 41 41]
char(b')
ans =
□□□□□□001abs(x)
□□□□□□001abs(x)
```

Коды до 32 не представляют никакие символы, поэтому функция **char** отобразила их условными квадратами.

Матрицу граничных условий можно создать по – другому

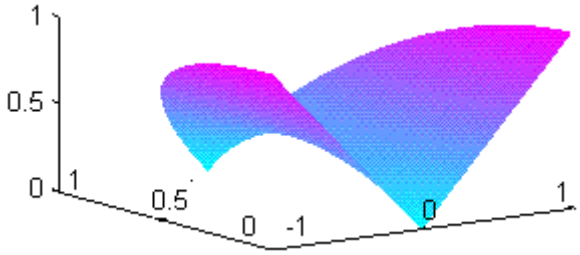
```
b=[1 1; 1 1; 1 1; 1 1; 1 1; 6 6; '0' '0'; '0' '0'; '1' '1';'a', 'a'; 'b' 'b'; 's' 's'; '(' '('; 'x' 'x';')' ')']
char(b(7:end,:)')
ans =
001abs(x)
001abs(x)
```

Эллиптическое уравнение $-div(c \cdot grad(u)) + a \cdot u = f$ принимает требуемую форму при `a=0, c=1, f=0`. Задаем коэффициенты и правую часть уравнения $\Delta u = 0$

```
a=0;          % коэффициент уравнения
```

c=1; `% коэффициент уравнения`
f = 0; `% Строка с формулой правой части уравнения`

Находим решение
u=assempde(b,p,e,t,c,a,f);
Строим график поверхности решения `u(x,y)`
pdesurf(p,t,u);



Пример 3. Решим ДУЧП следующего вида

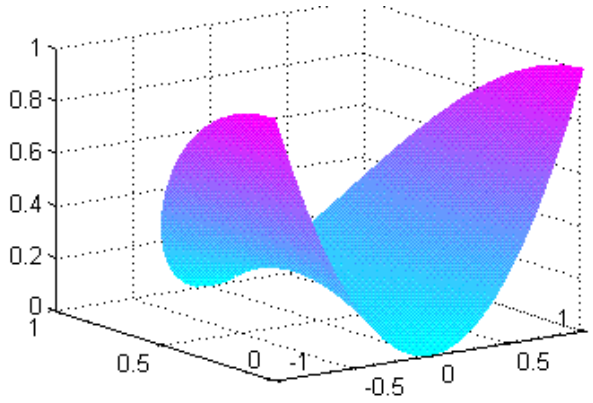
$$\nabla \cdot \left(\frac{1}{\sqrt{1+|\nabla u|^2}} \nabla u \right) = 0$$

в полукруге со значениями $u = x^2$ на границе $\partial\Omega$. Оно определяет форму минимальной поверхности, край которой находится на заданной высоте над контуром области.

Это нелинейное эллиптическое уравнение $-div(c \cdot grad(u)) + a \cdot u = f$ в котором коэффициент $c = \frac{1}{\sqrt{1+u_x'^2+u_y'^2}}$ является функцией производных u_x, u_y решения.

Для решения задачи создадим сценарий `t2.m`, использующий выше построенную функцию `halfcirclegeom` декомпозиционной геометрии полукруга.

```
% сценарий построения минимальной поверхности над полукругом
[p, e, t] = initmesh('halfcirclegeom');
[p, e, t]=refinemesh('halfcirclegeom',p,e,t);
% pdemesh(p,e,t); axis equal;

b=[1 1; 1 1; 1 1; 1 1; 1 1; 4 4; '0' '0'; '0' '0'; '1' '1';...
    'x', 'x'; '.' '.'; '^' '^'; '2' '2'];  % матрица граничных условий
a=0;      % коэффициент уравнения
f=0;      % Строка с формулой правой части уравнения
c='1./sqrt(1+ux.^2+uy.^2)';     % коэффициент уравнения
[u,res]=pdenonlin(b,p,e,t,c,a,f);
pdesurf(p,t,u); grid on;
```



Заметим, что для решения задачи мы выбрали солвер **pdenonlin**, который используется для решения нелинейных уравнений.

Пример 4. Решим ДУЧП минимальной поверхности, приведенное в предыдущем примере, в полукруге со значениями $1-|x|$ над прямолинейным участком границы и x^2-1 – над криволинейным. Для его решения создадим сценарий t3.m

```
% сценарий построения минимальной поверхности над полукругом
[p, e, t] = initmesh('halfcirclegeom');
[p,e,t]=refinemesh('halfcirclegeom',p,e,t);
pdemesh(p,e,t); axis equal;
b1=[1 1 1 1 1 6 48 48 49 120 46 94 50  45 49  0  0;
    1 1 1 1 1 8 48 48 49  49 45 97 98 115 40 120 41];
b=b1';

a=0;          % коэффициент уравнения
f=0;          % Строка с формулой правой части уравнения
c='1./sqrt(1+ux.^2+uy.^2)';    % коэффициент уравнения
[u,res]=pdenonlin(b,p,e,t,c,a,f,'Tol',0.01,'MaxIter',50);
pdesurf(p,t,u); grid on;


figure; axis([-1 1 0 1 -1 1]);
pdeplot(p,[],t,'xydata',u(:),'xystyle','off', ...
     'zdata',u(:),'zstyle','continuous','colorbar','off',...
     'mesh','on','xygrid','on');
colormap bone; grid on;
```

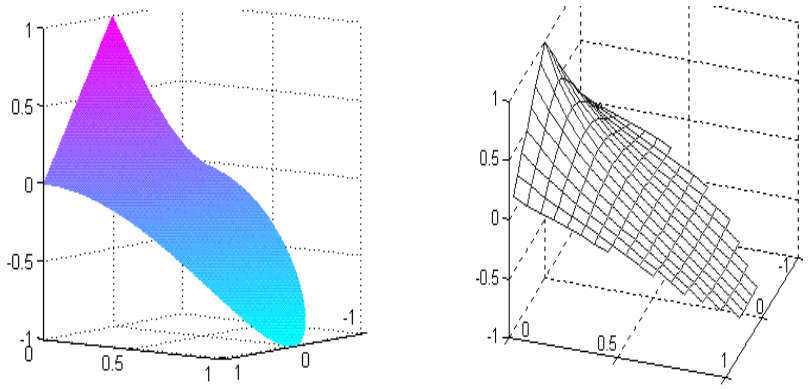
В конце первого столбца матрицы граничных условий мы поставили два нуля, чтобы выровнять длины столбцов.

char(b(7:end,:))'

```
ans =
001x.^2-1
0011-abs(x)
```

Обратите также внимание, что при вызове функции **pdenonlin** мы использовали опции, задающие точность вычислений (уменьшаем точность `'Tol',0.01`) и максимальное количество итераций (увеличиваем максимальное количество итераций `'MaxIter',50`). Без задания этих опций функция **pdenonlin** не справлялась со своей задачей и выводила сообщения об ошибках.

Сценарий завершается командами построения цветного и каркасного графиков минимальной поверхности.



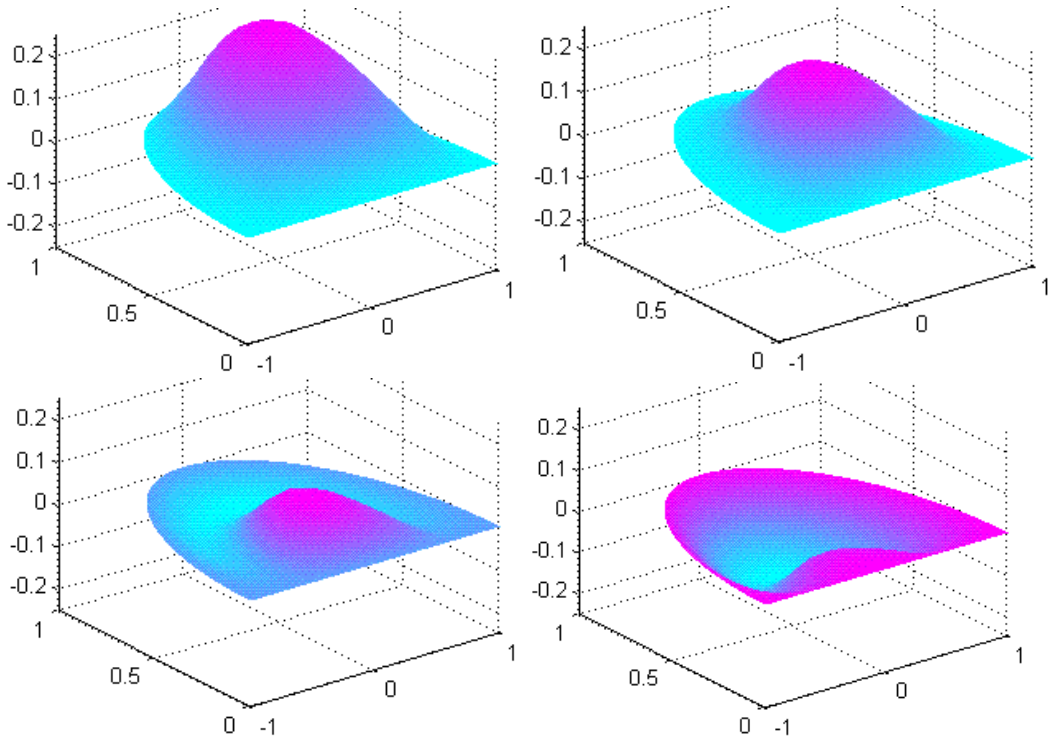
Пример 5. Решим задачу о колебании полукруглой мембраны закрепленной по контуру. Она состоит в решении уравнения $\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$ с граничными условиями $u\big|_{\partial\Omega} = 0$.

Общий вид гиперболического уравнения, решаемого пакетом PDE TOOLBOX, имеет вид $d\frac{\partial^2 u}{\partial t^2} - div(c\,grad(u)) + a\,u = f$. Чтобы оно имело требуемый вид, функции **hyperbolic** мы должны передать коэффициенты `d=1, c=1, a=0, f=0`.

Для решения задачи создадим сценарий `waveHalfCircle.m`, использующий выше построенную функцию `halfcirclegeom` декомпозиционной геометрии полукруга.

```
% сценарий построения поверхности колеблющейся мембраны над полукругом
[p, e, t] = initmesh('halfcirclegeom');
[p,e,t]=refinemesh('halfcirclegeom',p,e,t);
pdemesh(p,e,t); axis equal;

b1=[1 1 1 1 1 1 48 48 49 48 ;
    1 1 1 1 1 1 48 48 49  48 ];
b=b1';                  % матрица граничных условий

x=p(1,:)';
y=p(2,:)';

u0=(1-x.^2-y.^2).*y;     % начальное смещение точек мембраны
ut0= 0;                  % начальная скорость точек мембраны

n=21;
tlist=linspace(0,2,n);  % вектор моментов времени

u=hyperbolic(u0,ut0,tlist,b,p,e,t,1,0,0,1);
% строим кадр
pdesurf(p,t,u(:,3)); axis([-1 1 0 1 -0.25 0.25]);grid on;
```

В последней строке сценария мы строим поверхность мембраны `u(:,3)` в третий момент времени `tlist(3) = 0.2`. На следующем графике показаны формы мембраны в моменты времени `tlist([3, 4,5, 6])=[0.2 0.3 0.4 0.5]`.



4.1.3 ДУЧП с одной пространственной переменной

В MATLAB численное решение краевых задач для уравнений в частных производных с одной временной `t` и одной пространственной переменной x выполняется функцией **pdepe(…)**. Она "умеет решать" параболическое дифференциальное уравнение следующего вида

$$c(x,t,u,u_x)\cdot u_t = x^{-m}\frac{\partial}{\partial x}\left(x^m \cdot b(x,t,u,u_x)\right)+ s(x,t,u,u_x)$$

с граничными условиями

$$p(x_l,t,u)+ q(x_l,t)\cdot b(x_l,t,u,u_x)=0$$
$$p(x_r,t,u)+ q(x_r,t)\cdot b(x_r,t,u,u_x)=0$$

где x_l и x_r представляют левую и правую точку отрезка, и функция `u` вычисляется также в этих точках. Начальное условие должно иметь вид $u(0,x)=f(x)$. Отметим, что функция `b`, стоящая в уравнении и граничных условиях одна и та же. Функция **pdepe(…)** также "умеет решать" системы таких уравнений.

Функция вызывается командой

sol = pdepe(m, @pdefun, @icfun, @bcfun, xmesh, tspan, options),

где

- `m` параметр симметрии задачи: `m=0` – плоская симметрия, `m=1` – цилиндрическая симметрия, `m=2` – сферическая симметрия;
- `@pdefun` – дескриптор функции, определяющей коэффициенты `c, b, s` уравнения; заголовок функции следует оформлять следующим образом: `function [c,b,s] = pdefun(x,t,u,DuDx);`

- `@icfun` – дескриптор функции, определяющей начальное условие `f(x)`; заголовок функции следует оформлять следующим образом: `function u0 = icfun(x);`
- `@bcfun` – дескриптор функции, определяющей коэффициенты граничного условия; заголовок функции следует оформлять следующим образом:
  `function [pL,qL,pR,qR]=bcfun(xL,uL,xR,uR,t);`
- `xmesh` – вектор $[x_1, x_2, \ldots, x_n]$, определяющий точки, в которых будет вычисляться значения решения `u(t,x)` ($x_1<x_2<\ldots<x_n$);
- `tspan` – вектор $[t_0, t_1, \ldots, t_n]$ определяющий моменты времени, в которые будет вычисляться решение `u(t,x)` ($t_0<t_1<\ldots<t_n$, моментов времени должно быть три или более).
- `options` – необязательные аргументы, которые могут задавать точность определения решения, количество итераций и т.д.

Функция возвращает многомерный массив $u(t_j, x_k, i)$, представляющий аппроксимацию `i` – ой компоненты вектора решения системы уравнений $[u_1(t,x), u_2(t,x), \ldots, u_n(t,x)]$ в точках x_k=`xmesh(k)` в моменты времени t_j=`tspan(j)`. В случае одного уравнения этот массив представляет матрицу значений `u(j, k, 1)` одной функции – решения $u_1(t,x)$. Строками этой матрицы являются значения решения в некоторый момент времени. Для вычисления значения решения и ее производных в точках, не включенных в вектор `xmesh` можно использовать функцию **pdeval**.

Опциями функции **pdepe** могут быть некоторые из опций, используемых при решении обыкновенных дифференциальных уравнений, – `RelTol`, `NarmControl`, `InitialStep` и `MaxStep`. Их задание выполняется функцией **odeset**. Значения этих опций по умолчанию удовлетворительно для большинства случаев.

Использование функции **pdepe** рассмотрим на примерах.

Пример 1. Решим на отрезке [0, 1] нестационарное одномерное уравнение теплопроводности $\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}$ с граничными условиями $u(t,0) = 0$, $u'_x(t,1) = 0$ и начальным условием $u(0,x) = 0.5 - \left| 0.5 - x \right|$.

В нашем случае `m=0` и коэффициенты в уравнении общего вида должны быть равны

$$c(x,t,u,u_x) = 1,\ \ b(x,t,u,u_x) = u_x,\ \ s(x,t,u,u_x) = 0.$$

Коэффициенты граничных условий для нашего примера имеют вид

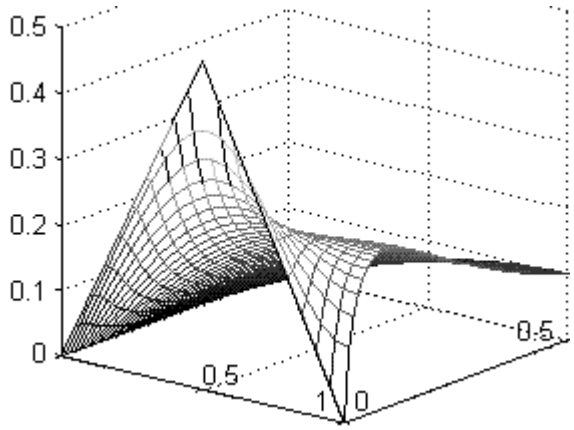
$$p(0,t,u) = u,\ \ q(0,t) = 0,\ \ p(1,t,u) = 0,\ \ q(1,t) = 1.$$

Перед вызовом функции **pdepe** следует создать функции, вычисляющие коэффициенты уравнения и граничных условий. Их нельзя помещать в файл

– сценарий, поэтому для решения задачи удобно создавать m – функцию `pdex1.m` (без аргументов), в конец которой можно поместить код подфункций для вычисления требуемых коэффициентов.

```
function pdex1
% Решение одномерного уравнения теплопроводности с нулевой
% температурой на  левом краю и условием теплоизолированности на правом.

m = 0;                        % плоская задача
x = linspace(0,1,21);         % вектор точек
t = linspace(0,0.5,51);       % вектор моментов времени

sol = pdepe(m,@koefpde,@initpde,@boundpde,x,t); % Решаем ДУЧП
u = sol(:,:,1);    % скалярное уравнение => функция решения одна

% график поверхности решения
figure;
[X,T] = meshgrid(x,t);
mesh(X,T,u);
colormap Gray;
title('Численное решение на 20 точках.')
xlabel('Расстояние x');
ylabel('Время t');

% график решения в заданный момент времени
figure;
numfr=21;            % номер момента времени
plot(x,u(numfr,:)); % график решения в этот момент времени
s1=sprintf('Температура при t = %0.2g',t(numfr));
s2=sprintf('u(x,%0.2g)',t(numfr));
title(s1);
xlabel('Расстояние x');
ylabel(s2);

% простейшая анимация
fig = plot(x,u(1,:),'erase','xor');
for k=2:length(t)
    set(fig,'xdata',x,'ydata',u(k,:));
    pause(.1);
end

% --------------------------------------------------------------
% функции для вычисления коэффициентов и начального условия
% --------------------------------------------------------------
function [c,f,s] = koefpde(x,t,u,DuDx)
% коэффициенты уравнения
c = 1;
f = DuDx;
s = 0;
% --------------------------------------------------------------
function u0 = initpde(x)
% начальное значение
u0 = 0.5-abs(x-0.5);
% --------------------------------------------------------------
function [pl,ql,pr,qr] = boundpde(xl,ul,xr,ur,t)
% коэффициенты граничных условий
pl = ul;
ql = 0;
pr = 0;
qr = 1;
```

График функции `u(t,x)` показан на следующем рисунке.



Пример 2. Рассмотрим пример решения нелинейной параболической системы ДУЧП с одной пространственной переменной

$$\frac{\partial u_1}{\partial t} = \frac{\partial^2 u_1}{\partial x^2} + u_1 \cdot (1 - u_1 - u_2)$$
$$\frac{\partial u_2}{\partial t} = \frac{\partial^2 u_2}{\partial x^2} + u_2 \cdot (1 - u_1 - u_2)$$

с граничными условиями

$$u'_{1x}(t,0) = 0;\ u_1(t,1) = 1$$
$$u_2(t,0) = 0;\ u'_{2x}(t,1) = 0$$

и начальными условиями

$$u_1(0,x) = x^2\ ;\ u_2(0,x) = x(x-2)$$

Общий вид системы двух параболических уравнений следующий

$$c_1(x,t,u,u_x) \cdot u'_{1t} = x^{-m} \frac{\partial}{\partial x}\left(x^m \cdot b_1(x,t,u,u_x)\right) + s_1(x,t,u,u_x)$$
$$c_2(x,t,u,u_x) \cdot u'_{2t} = x^{-m} \frac{\partial}{\partial x}\left(x^m \cdot b_2(x,t,u,u_x)\right) + s_2(x,t,u,u_x)$$

Для нашего примера `m=0` и коэффициенты будут равны

$$c = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix};\ b = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} u'_{1x} \\ u'_{2x} \end{pmatrix};\ s = \begin{pmatrix} s_1 \\ s_2 \end{pmatrix} = \begin{pmatrix} u_1(1-u_1-u_2) \\ u_2(1-u_1-u_2) \end{pmatrix};$$

которые мы будем создавать в теле функции `eqnsys.m`

```
function [c,b,s]=eqnsys(x,t,u,DuDx)
% Файл создания коэффициентов системы двух PDE одной временной
% и одной пространственной переменной
c=[1; 1];
b=[1; 1].*DuDx;
s=[u(1).*(1-u(1)-u(2)); u(2).*(1-u(1)-u(2))];
```

Общий вид граничных условий для системы двух параболических уравнений следующий

$$p_1(x_l,t,u) + q_1(x_l,t) \cdot b_1(x_l,t,u,u'_x) = 0$$
$$p_1(x_r,t,u) + q_1(x_r,t) \cdot b_1(x_r,t,u,u'_x) = 0$$

$$p_2(x_l,t,u)+q_2(x_l,t)\cdot b_2(x_l,t,u,u'_x)=0$$
$$p_2(x_r,t,u)+q_2(x_r,t)\cdot b_2(x_r,t,u,u'_x)=0$$

Для нашего примера имеем

$$p(0,t,u)=\begin{pmatrix}p_1\\p_2\end{pmatrix}=\begin{pmatrix}0\\u_2\end{pmatrix};\quad q(0,t)=\begin{pmatrix}q_1\\q_2\end{pmatrix}=\begin{pmatrix}1\\0\end{pmatrix};$$

$$p(1,t,u)=\begin{pmatrix}p_1\\p_2\end{pmatrix}=\begin{pmatrix}u_1-1\\0\end{pmatrix};\quad q(1,t)=\begin{pmatrix}q_1\\q_2\end{pmatrix}=\begin{pmatrix}0\\1\end{pmatrix};$$

Их мы будем задавать функцией `boundsys.m`

```
function [pl,ql,pr,qr]=boundsys(xl,ul,xr,ur,t)
% функция создания коэффициентов граничного условия PDE системы
pl=[0;ul(2)];
ql=[1;0];
pr=[ur(1)-1;0];
qr=[0;1];
```

Для начальных условий $u_1(0,x)=x^2$; $u_2(0,x)=x(x-2)$ создадим функцию `initsys.m`

```
function value=initsys(x)
% начальное условие для PDE системы
value=[x^2;x.*(x-2)];
```

Теперь создадим сценарий `solvesys.m`, который будет решать систему и строить анимацию решения, т.е. строить график пары функций u_1`(x,t)`, u_2`(x, t)` в различные моменты времени.

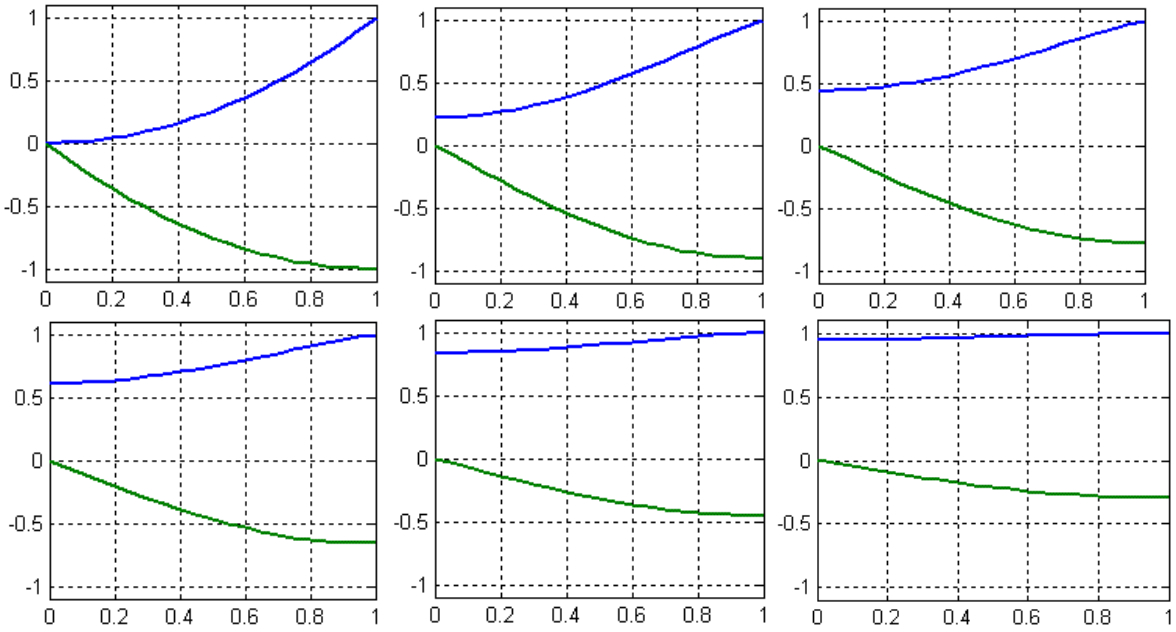
```
% сценарий решения PDE системы
m=0;
n=21;
x=linspace(0,1,n);
t=linspace(0,1,n);
sol=pdepe(m,@eqnsys,@initsys,@boundsys,x,t);
u1=sol(:,:,1);
u2=sol(:,:,2);

newplot;                              % создаем новое графическое окно
clear M;
for i=1:n,
    plot(x,u1(i,:),x,u2(i,:),'LineWidth',2);      % рисуем кадр
    axis([0 1 -1.1 1.1]);    % приводим кадр к общему масштабу
    grid on;
    title('u1(x,t), u2(x,t)');
    M(i)=getframe;                    % запоминаем кадр в массив
end
movie(M,4);                           % проигрываем массив кадров 4 раза
```

Для построения кадров по-отдельности (чтобы их можно было сохранить в отдельные графические файлы) создадим следующую функцию

```
function framesys2(nf,x,u1,u2)
% построение кадра решения PDE системы
% nf - номер момента времени (кадра)
plot(x,u1(nf,:),x,u2(nf,:),'LineWidth',2);
axis([0 1 -1.1 1.1]);
grid on;
```

Первый аргумент – номер кадра, который мы хотим построить, остальные – переменные, созданные в сценарии. Их необходимо либо передавать аргументами в тело нашей функции, либо объявлять как глобальные. Мы выбрали первый способ.

На следующем рисунке приведены графики функций u_1, u_2 в моменты времени `t = 0, 0.1, 0.2, 0.3, 0.5, 0.7.`



ЗАКЛЮЧИТЕЛЬНЫЕ ЗАМЕЧАНИЯ.

В главе 4.1 мы привели основные сведения, необходимые для решения некоторых ДУЧП. Однако пакет PDE TOOLBOX достаточно сложен и его возможности шире тех примеров, которые рассмотрены здесь. Перечислим некоторые из задач, решение которых возможно средствами пакета

- стационарные и нестационарные задачи теплопроводности;
- задачи диффузии;
- электростатические задачи;
- двумерные задачи теории упругости;
- задачи дифракции;
- распространение акустических и электромагнитных волн;
- определение собственных колебаний конструкций;
- ламинарное течение жидкости и газа;
- задачи теории пластин и оболочек.

Для решения той или иной прикладной задачи мы должны привести уравнения, описывающие явление, к виду, используемому одним из солверов пакета. Кроме того, читатель, владеющий навыками программирования в MATLAB, может легко расширить приведенный список, создавая собственные функции для решения задач из других областей техники и технологии.