Доля П.Г.
Харьковский Национальный Университет
механико – математический факультет
2013 г.

Методы обработки изображений.

В настоящем пособии излагаются методы, которые используются при обработке изображений. Перед вами часть, которая знакомит с алгебраическими методами обработки монохромных цифровых изображений.

Часть 1

Алгебраические методы обработки монохромных цифровых изображений.

Оглавление

1. Введение.

1.1 Основные задачи по обработке изображений

Монохромное изображение можно определить как двумерную функцию $f(x,y)$, где x и y — координаты на плоскости изображения, а амплитуда `f` называется интенсивностью или яркостью изображения в точке с этими координатами. Словосочетание уровень серого используется для обозначения яркости монохромного изображения. Цветные изображения формируются комбинацией нескольких монохромных изображений. Например, в цветовой системе RGB цветное изображение строится из трех отдельных монохромных компонент (красной, зеленой и синей) и может быть представлено вектором из трех интенсивностей $\left(f_r(x,y), f_g(x,y), f_b(x,y)\right)$. По этой причине многие методы и приемы,

разработанные для монохромных изображений, могут быть распространены на цветные изображения путем последовательной обработки трех монохромных компонент.

Первичное изображение имеет непрерывные x - и y - координаты, а также непрерывную амплитуду f. Преобразование его в цифровую форму требует представления координат и значений амплитуды некоторыми дискретными отсчетами. Представление координат конечным множеством отсчетов называется дискретизацией, а представление амплитуды значениями из конечного набора называется квантованием. Если координаты x и y, а также величины амплитуды f выбираются из фиксированных конечных наборов элементов (дискретных величин), то изображение называется цифровым изображением.

Цифровая обработка изображений имеет весьма широкие сферы применения, включая зрительные образы. Однако в отличие от людей, которые способны воспринимать лишь электромагнитное световое излучение видимого диапазона, машинная обработка изображений покрывает практически весь спектр электромагнитных и других волн. Например, это ультразвуковые изображения или изображения, полученные в электронной микроскопии.

Не существует общепризнанной границы, которая разделяла бы область обработки изображений и другие смежные дисциплины, такие как анализ изображений или машинное зрение. Иногда такое разграничение делается по принципу, что обработка изображений характеризуется присутствием изображений на входе и выходе системы. Однако такое определение представляется нам неоправданно ограничительным и искусственным.

Во всем многообразии задач от обработки изображений до машинного зрения нет четких границ, однако здесь можно выделить компьютеризованные процессы низкого, среднего и высокого уровня. Процессы низкого уровня включают лишь примитивные операции над изображениями типа уменьшение шума, повышение контрастности или улучшение резкости. Они характеризуются тем, что на вход и выход поступают изображения. Процессы среднего уровня связаны с такими задачами, как сегментация (разделение изображений на области и выделение в них объектов), описание объектов и их сжатие для придания им удобной формы при дальнейшей компьютерной обработке, а также классификация (распознавание) выделенных объектов. В среднеуровневых процессах на входе имеются изображения, а на выход поступают атрибуты и признаки, извлеченные из этих изображений, например, границы, контуры и другие отличительные признаки объектов, которые также являются изображениями. Наконец, процессы высокого уровня занимаются «осмыслением» множества распознанных объектов, как это делается в анализе изображений.

Имея в виду перечисленные выше замечания, видно, что естественным полем пересечения и перехода от обработки изображений к их анализу является область распознавания отдельных фигур или объектов на

изображениях. Таким образом, то, что называется цифровой обработкой изображений, связано с процессами, имеющими изображения на входе и на выходе, а также с процессами извлечения определенных изображений. Ближайшими к обработке изображений задачами из анализа изображений являются проблемы распознавания отдельных объектов. В качестве иллюстрации, проясняющей эти концепции, можно привести задачу автоматического анализа печатного или рукописного текста.

Для понимания обработки изображений важно решать практические задачи, а для этого надо иметь готовый компьютерный инструмент способный проиллюстрировать применение тех или иных методов. В нашем спецкурсе мы будем использовать MatLab. Он является средой для выполнения технических и научных вычислений. В нем интегрированы вычисления, визуализация и программирование в удобной пользовательской среде, в которой задачи и их решения выражаются с помощью привычных математических обозначений. Базовым элементом MATLAB является массив элементов (матрица), который не требует задания фиксированной размерности. Это позволяет легко формулировать условия и решения многих вычислительных задач, которым требуется матричное представление объектов. Система MATLAB имеет расширения в виде наборов специализированных программ, которые по-английски называются toolbox (набор инструментов). Пакет Image Processing Toolbox (IPT) состоит из функций MATLAB (они называются M-функции или M-файлы), которые расширяют возможности стандартной среды MATLAB для решения задач цифровой обработки изображений. Используя функции этого пакета, мы будем иллюстрировать приемы по обработке изображений.

1.2 Полезные графические функции MATLAB

Рассмотрим некоторые наиболее важные для нашего спецкурса функции MATLAB. Создадим матрицу 5 x 5

```
>>A=[1 2 3 0 0; 4 5 6 0 0; 7 8 9 0 0; 0 0 0 0 0; 0 0 0 0 0]
A =
     1     2     3     0     0
     4     5     6     0     0
     7     8     9     0     0
     0     0     0     0     0
     0     0     0     0     0
```

Чтобы ее увидеть как изображение используем команду

```
>>imshow(A);  % рисунок слева
```

Мы показали небольшую часть графического окна MATLAB.

Функция `mat2gray(A)` переводит элементы матрицы `A` к диапазону [0 1].

```
>>B=mat2gray(A)
B =
```

```
    0.1111    0.2222    0.3333         0         0
    0.4444    0.5556    0.6667         0         0
    0.7778    0.8889    1.0000         0         0
         0         0         0         0         0
         0         0         0         0         0
```

А функция `im2uint8(B)` преобразует матрицу-аргумент к диапазону [0 255].

```
>>C=im2uint8(B)
C =
    28   57   85    0    0
   113  142  170    0    0
   198  227  255    0    0
     0    0    0    0    0
     0    0    0    0    0
>>imshow(C);        % предыдущий рисунок справа
```

Посмотрите также в окне Workspace на тип матриц: `A – double, B – double, C – uint8`.

Есть также функция `im2uint16`, которая переводит величины в диапазон [0 65535].

Для матрицы A, созданной нами выше,

```
>>A=[1 2 3 0 0; 4 5 6 0 0; 7 8 9 0 0; 0 0 0 0 0; 0 0 0 0 0];
```

команда

```
>>image(A);
```

создает большое изображение (с громадными пикселями) в графическом окне.

Для загрузки (чтения) изображений в рабочее пространство MATLAB используется функция `imread` со следующим синтаксисом:

```
>>imread('filename')
```

Здесь `filename` — это строка символов, образующих полное имя загружаемого файла изображения (включая любое расширение). Например, командная строка

```
>>f=imread('football.jpg');
```

присваивает изображение формата JPEG с именем «`football.jpg`» матричной переменной *f*. Заметим, что символ ‘ (апостроф) используется в качестве ограничителя символьной строки. Точка с запятой в конце командной строки означает инструкцию системе MATLAB не отображать вывод в ее командное окно. Если точка с запятой отсутствует, то MATLAB отображает результат (вывод) выполнения операций в командной строке. Символ приглашения >> обозначает начало командной строки в окне команд MATLAB. Этот символ в примерах нашего конспекта мы писать больше не будем.

Если в имя файла не включена информация о пути к данному файлу, то файл `filename` ищется в текущей папке. А если его там нет, то выполняется поиск данного файла во всех папках, пути к которым указаны в путях поиска MATLAB. В данном пособии мы предполагаем, что все графические файлы собраны в одном каталоге, на который в MATLAB установлены пути поиска.

Функция `size(f)` возвращает размер изображения, т. е. число строк и столбцов массива, представляющего изображение:

```
size(f)   % f содержит изображение football.jpg
ans = 256 320  3
```
Эта функция будет особенно полезной при автоматическом определении размера изображения, которое делается операцией:
```
[M,N,K] = size(f);
```
При такой записи переменной M будет присвоено число строк изображения, а переменной N — число столбцов, K – число цветовых плоскостей.

Функция whos сообщает дополнительную информацию о массиве. Например, строка
```
whos f
```
выдает следующий результат:
```
Name       Size              Bytes   Class
 f        256x320x3          245760  uint8 array
Grand total is 245760 elements using 245760 bytes
```
Функция numel(f) возврщает число элементов массива f, т.е. число пикселов изображения
```
numel(f)
ans =
      245760
```
Как мы говорили выше, загруженное изображения можно вывести на дисплей компьютера с помощью функции imshow, которая имеет следующий синтаксис:
```
imshow(f);
```
или
```
imshow(f, G),
```
где f — это матрица изображения, а G — это число уровней яркости, используемое при отображении этого изображения. Если аргумент G опущен, то по умолчанию принимается 256 уровней яркости (то которое получено из графического файла). Команда
```
imshow(f,[low high])
```
означает, что все пикселы со значением не больше числа low надо показывать черными, а все пикселы со значениями не меньше числа high — белыми. Все значения расположенные между low и high показываются с промежуточной яркостью с использованием числа уровней принятому по умолчанию. Наконец, запись в командной строке
```
imshow(f,[ ] )
```
задает для переменной low минимальное значение массива f, а переменной high присваивается его максимальное значение. Такая форма функции imshow бывает полезной при показе изображений, имеющих узкий динамический диапазон значений пикселов, или когда среди них имеются положительные и отрицательные значения.

В настоящем пособии мы будем изучать методы обработки изображений. Но после обработки оно должно быть сохранено в графический файл. Это можно выполнить командой
```
imwrite(A,filename,fmt),
```

которая сохраняет образ A в файл, заданный именем filename в графическом формате, определенном аргументом fmt. Например, для изображения футбольного мяча, хранящегося в файле 'football.jpg', следующие команды создадут графический файл того же изображения, но в формате bmp

```
f=imread('football.jpg');  % обычно следует обработка изображения f
imwrite(f,'football.bmp','bmp');
```

Теперь поговорим немного подробнее о графических файлах MATLAB.

Одна из двух форм графических файлов в MATLAB имеет вид матрицы с элементами, обычно представимыми целыми числами, которые являются индексами (номерами) цветов в матрице называемой картой или палитрой цветов (colormap). Рассмотрим для примера рисунок, который находится в графическом файле Pogorelov32x32.bmp, размером 32 на 32 пикселя.

Команда

```
A=imread('Pogorelov32x32.bmp')
```

читает и печатает матрицу A размером 32 x 32. Значения элементов матрицы являются номерами цветов соответствующих пикселей. Вот кусочек этой матрицы

```
  15  15  15  15   0   0   0   0   0   1   6 …
  15   0   0   6   0   7   7   8   8   8   8 …
  15  15   0   0   0   1   7   8   7   8   8 …
  15   7   0   2   1   7   7   8   8   8   8 …
  15   5   0   7   0   7   7   8   8   8   8 …
  ……..……..……..……..……..……..……..……..……..……..……..
```

Команда

```
image(A);
```

по этой матрице рисует картинку в графическом окне (рисунок слева).

Рисунок не очень хорош и для получения приемлемого изображения нам потребуется изменить параметры его отображения.

Когда матрица содержит целые положительные числа, то ее элементы интерпретируются как индексы в карте цветов. Функция image(A) строит ее образ в текущей карте – каждому элементу матрицы будет соответствовать некоторый прямоугольник на рисунке, закрашенный цветом, номер которого соответствует строке матрицы цветов (палитра). Имеются также дополнительные аргументы, управляющие работой функции image.

Полученным образом в графическом окне, можно управлять, используя обычные функции MatLab, управляющие графикой. В частности, в нашем примере рисунок вытянут по горизонтали. Команда

```
axis equal;
```

выравнивает горизонтальные и вертикальные размеры пикселей в графическом окне. Она устанавливает коэффициент сжатия изображения одинаковым по всем осям (см. предыдущий рисунок справа). Тот же результат дает команда

```
axis image;
```

но охватывающий изображение прямоугольник в графическом окне деформируется под размер изображения.

Цвета точек берутся из текущей карты цветов. Карта цветов (палитра) – это матрица, которая содержит 3 столбца и несколько (может быть много) строк. Например, команда

```
gray(5)
```

создает следующую матрицу, которая может быть использована как палитра цветов

```
         0         0         0
    0.2500    0.2500    0.2500
    0.5000    0.5000    0.5000
    0.7500    0.7500    0.7500
    1.0000    1.0000    1.0000
```

Карта цветов `colormap` является $m \times 3$ матрицей вещественных чисел диапазон изменения которых от 0 до 1. Каждая ее строка является RGB вектором, определяющим один цвет. Строка карты цветов с номером k определяет $k-$й цвет в формате $[r(k)\ g(k)\ b(k)]$ с соответствующими интенсивностями красного `r(k)`, зеленого `g(k)` и синего `b(k)` цветов.

Мы видели, что матрица `A`, прочитанная из файла, содержит числа от 0 до 15. Для того, чтобы убедиться в этом достаточно выполнить команды

```
min(min(A))
ans =    0
max(max(A))
ans =   15
```

Команды `max` и `min` определяют наибольшее и наименьшее значение матрицы по одной из ее размерностей. Например, однократное применение команды `max(A)` возвращает вектор из 32 чисел являющихся максимальными значениями по соответствующим столбцам матрицы `A`. Для определения максимального значения всей матрицы нужно еще раз применить функцию `max` к результату ее первого применения.

Поскольку матрица A содержит числа от 0 до 15, то разумно создать черно – белую палитру с 16 оттенками серого. Команда

```
colormap(gray(16));
```

преобразует рисунок к следующему виду

Обратите внимание на тип элементов матрицы A в окне Workspace – uint8 (беззнаковые 8 – битные целые могут изменяться в диапазоне от 0 до 255). Если выполнить команду

```
B=A/15;
```

то операция деления целых чисел даст только 0 или 1, а тип элементов матрицы B не изменится (целое разделить на целое дает целое). Это позволяет создать черно – белый рисунок. Для этого создадим палитру из двух цветов – черного (ему соответствует строка 0, 0, 0) и белого (ему соответствует строка 1, 1, 1).

```
cm=[0 0 0;1 1 1];   % матрица палитры (два цвета – черный
                                  и белый)
image(B);
axis image; colormap(cm);
```

```
>> cm
cm =
     0     0     0
     1     1     1
```

Команда

```
colormap('default')
```

устанавливает карту цветов по – умолчанию.

Графический файл мог сразу быть монохромным. Например, автор создал в Paint-е черно-белую картинку 32 x 32 пикселя AuthorMono.bmp

Команда

```
P=imread('AuthorMono.bmp')
```

прочитала и отпечатала матрицу 32 x 32 пикселя, составленную только из нулей или единиц. Команда

```
image(P);axis image; colormap('default');
```

нарисовала очень темную картинку. Но команды

```
cm=[0 0 0;1 1 1];colormap(cm);
```

преобразуют картинку к виду, который показан выше.

Чтобы узнать формат графического файла можно использовать команду

```
info = imfinfo('Pogorelov32x32.bmp')
```

Она возвращает структуру с большим количеством полей, содержащую различную информацию о графическом файле. Здесь мы приводим некоторые поля этой структуры

```
info =
              Filename: 'Pogorelov32x32.bmp'
              FileSize: 630
                Format: 'bmp'
```

```
        FormatVersion: 'Version 3 (Microsoft Windows 3.x)'
                Width: 32
               Height: 32
             BitDepth: 4
            ColorType: 'indexed'
    NumColormapEntries: 16
             Colormap: [16x3 double]
           NumPlanes: 1

          ..................
```

Обратите внимание на строку с параметром `ColorType`, в которой указывается тип графического файла. В настоящем пособии мы с вами встретим значения этого параметра равными `'indexed'`, `'grayscale'`, `'truecolor'`. Сейчас мы говорим только о графических файлах типа `'indexed'`.

Команда

```
[X,map] = imread(filename)
```

читает графический файл в матрицу `X`, а его карту цветов в матрицу `map`, элементы которой масштабируются к диапазону [0, 1].

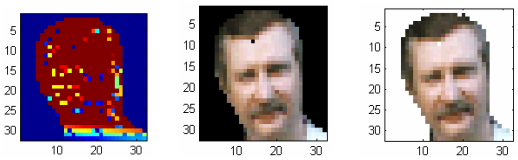
```
[A,m] = imread('Pogorelov32x32.bmp');
image(A);
colormap(m);
```

Команда

```
[A,map,alpha] = imread(...)
```

в третьем параметре возвращает маску (матрицу из логических нулей и единиц), которая используется для определения информации о прозрачности точек образа (какие точки образа отображать на экране, а какие нет; не отображать те, для которых элементы `alpha` равны 1). Например, этот параметр полезен для рисования образа файла иконки.

```
[A,B,C]=imread('Author.ico'); % A содержит числа от 0 до 255.
image(A); axis image;           % левый рисунок
colormap(B);                    % средний рисунок
```



Теперь сделаем фон рисунка белым.

```
B2=B;                   % копируем палитру в B2
B2(256,:)=[1 1 1];      % последний цвет палитры сделать белым цветом
D = ones(size(A)) * (length(B2)-1); %создать матрицу с элементами 255
D(C == 0) = A(C == 0);
image(uint8(D)), colormap(B2);  %правый рисунок
axis image;
```

Поясним работу строки `D(C==0)=A(C==0)`.

Если посмотреть в окне Workspace, то видно, что массив `C` является логическим. И при этом он имеет такой же размер, что и массив `A`. Здесь используется логическое индексирование в форме

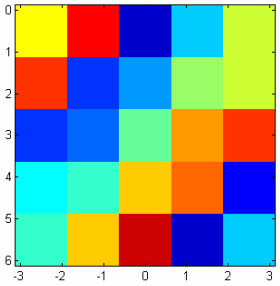
```
имя_матрицы(логический массив)
```

где логический массив должен иметь тот же размер, что и матрица. Результатом этой операции логического индексирования является вектор, составленный из элементов исходной матрицы, для которых в логическом

массиве соответствующие элементы равны логической единице. Этот вектор может стоять как в правой части оператора присваивания, так и в левой. Вначале мы создали `double` массив `D` из чисел 255, а затем создали вектора одинаковой длины `D(C==0)` и `A(C==0)`. Второй вектор содержит только элементы матрицы `A`, для которых в матрице `C` элементы равны нулю. Это значит, что соответствующие точки образа должны быть нарисованы. Значения этого вектора `A(C==0)` присваиваются элементам вектора `D(C==0)`, т.е. соответствующим элементам матрицы `D`. Если в матрице `C` элемент был 1, то соответствующий элемент матрицы `D` не меняется, т.е. остается равным 255. В результате элементы матрицы `A`, отвечающие отображаемым точкам, скопировались в матрицу `D`, а отвечающие неотображаемым точкам остались в матрице `D` равными 255. Кроме того, нам потребовалось заменить последнюю (256 – ю) строку матрицы палитры, чтобы она соответствовала белому цвету. Поскольку функция `image` отображает только матрицы с типом элементов `uint8` (диапазон изменения чисел от 0 до 255), то нам также потребовалось преобразование типа вещественной матрицы `D` в тип `uint8`.

Команда `image(x,y,C)`, где `x` и `y` являются двухэлементными векторами, которые определяют диапазон изменения координат `x` и `y`, рисует такой же образ, что и команда `image(C)`. Это бывает полезно, например, в случае, когда вы желаете, чтобы метки осей соответствовали реальным физическим координатам графического образа.

```
A = magic(5);
x = [-2.5 2.5]; y = [0.5 5.5];
image(x,y,A), axis image, colormap(jet(25))
```



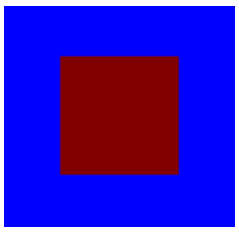
Функция `imshow(M)` тоже строит графический образ полутоновой (т.е. одноцветной с различными градациями яркости) матрицы `M` элементы которой имеют тип `double` (или `uint8`, `uint16`). При этом каждой точке матрице соответствует один пиксель в графическом окне. Вариантов использования этой функции очень много. Она является базовой графической функцией пакета `Image Processing Toolbox`. С различными вариантами ее использования следует ознакомиться по справочной системе.

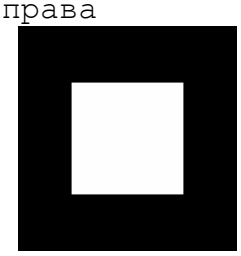
```
I = zeros(4,4) % генерирование нулевой матрицы размером 4 x 4
I =   0     0     0     0
      0     0     0     0
      0     0     0     0
      0     0     0     0
I(2:3, 2:3) = 1     % изменение значений матрицы
```

```
I =   0     0     0     0
      0     1     1     0
      0     1     1     0
      0     0     0     0
imshow(I);            % построение образа матрицы
```

Здесь единицы в матрице `I` соответствуют белым точкам рисунка, а нули – черным. Но точек мало, поэтому рисунок маленький. Если бы мы использовали функцию `image`, то нам надо было бы выполнить преобразование типа, например, так `image(uint8(I))`, и при этом образ матрицы был бы построен по размеру графического окна. Увеличим размер рисунка, увеличив размер матрицы

```
I = zeros(100,100);
I(25:75, 25:75) = 1;
colormap('default');
imshow(I);     % следующий рисунок слева
colormap(hot); % следующий рисунок справа
```





Слово `hot` представляет имя предопределенной матрицы палитры. Имена готовых матриц палитр можно найти в справочной системе. Вот еще пример использования функции `imshow`.

```
moon = imread('moon.tif');
imshow(moon);
```

или можно сразу использовать имя файла

```
imshow('moon.tif');
```

Команда

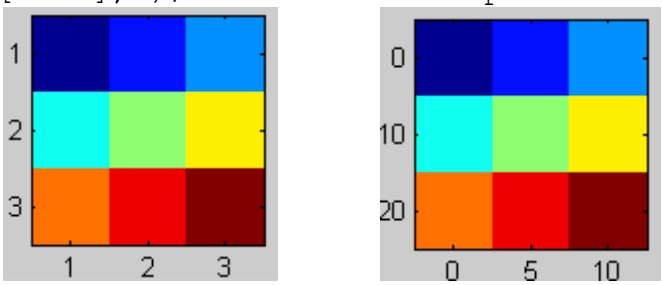
```
imshow(I,[low high])
```

рисует образ матрицы `I` в серых оттенках с диапазоном изменения значений `I` равным `[low high]`. Значение `low` и меньшие рисуются черным цветом, значение `high` и большие рисуются белым. Значения между `low` и `high` рисуются промежуточными оттенками серого, используя уровни серого определяемые по – умолчанию. Если используется пустая матрица `[]` вместо `[low high]`, то `imshow` использует `[min(I(:))max(I(:))]`, т.е. минимальное значение в `I` отображается черным, а максимальное – белым цветами.

Угадывать палитру или использовать палитру графического файла не всегда удобно. Поэтому в MatLab есть функция `imagesc`, которая масштабирует матрицу под текущую палитру и затем рисует ее образ.

Команда `imagesc(A)` отображает матрицу как графический образ. Каждый элемент матрицы соответствует прямоугольному участку образа. Значения элементов являются индексами строк текущей матрицы палитры, в которой тройка чисел каждой строки в формате RGB определяет цвет. Команда `imagesc(x,y,A)` отображает матрицу `A` в виде графического образа на координатной сетке с диапазоном определяемым векторами `x` и `y`. Например

```
A=[1 2 3; 4 5 6; 7 8 9];
imagesc(A); colormap('default');  % изображение слева
imagesc([0 10],[0 20],A);          % изображение справа
```

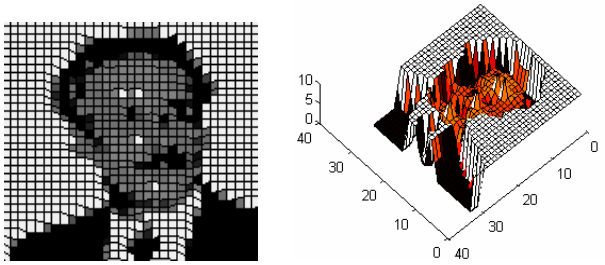


Иногда желательно отобразить матрицу как поверхность над прямоугольной областью. Это можно сделать с помощью функции `surface(Z)`, где матрица `Z` содержит вещественные числа, которые будут интерпретироваться как высота поверхности над плоскостью XY. Но матрицы, прочитанные из графического файла, содержали только целые числа. Здесь нам может потребоваться преобразование типа. Например, команда

```
A = imread('Pogorelov32x32.bmp');
C = double(A)/16;
```

создаст вещественную матрицу `double(A)`, каждый элемент которой будет вещественным числом, которые после деления на 16 создадут матрицу `C` с действительными элементами в диапазоне от 0 до 1.

```
C=double(A)/16;
surface(C);   %  рисует поверхность матрицы с double
                 элементами,
surface(C*10); colormap(hot);
```



Аналогично работает команды `surf(Z)` и `surfc(Z)` которые создают 3 – х мерный график поверхности по `z` координатам точек, заданным в матрице `Z` используя `x=1:n` и `y=1:m`, где `[m,n]=size(Z)`. Высоты точек

поверхности Z являются вещественными числами, а цвет поверхности пропорционален высоте точек.

```
A = imread('Pogorelov32x32.bmp')
C=double(A)/16;
surfc(C);
colormap(gray(17));
```

Полезной может быть также функция построения каркасного изображения поверхности. Один из наиболее часто используемых ее синтаксисов следующий

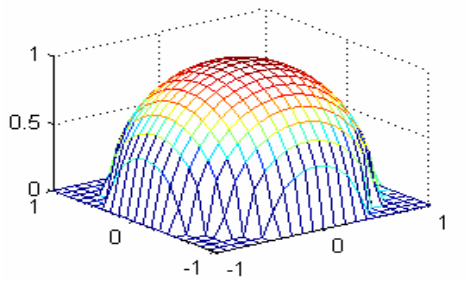
```
mesh(X,Y,Z);
```

где матрицы X, Y, Z должны быть одинакового размера. График поверхности строится над прямоугольной областью плоскости *XY*, которая разбивается сеткой с размером таким же, как у матриц. В узлах сетки значения функции равны соответствующим значениям элементов матрицы Z. Матрицы X и Y содержат *x* и *y* координаты узлов сетки. В следующем примере мы строим единичную полусферу по уравнению

$$z = \begin{cases} \sqrt{1 - x^2 - y^2}, & x^2 + y^2 \le 1 \\ 0, & x^2 + y^2 > 1 \end{cases},$$

где $-1 \le x \le 1$, $-1 \le y \le 1$.

```
[X Y]=meshgrid(-1:0.1:1);
XY=X.^2+Y.^2;
C=XY<=1;
XYZ=zeros(size(XY));   % нулевая матрица
XYZ(C)=1-XY(C);        % логическое индексирование
Z=sqrt(XYZ);
mesh(X,Y,Z);
```



Здесь мы использовали логическое индексирование так же, как описано выше в этом параграфе.

Перечислим функции, которые рассмотрены в этом параграфе

`imread('файл',...)`	чтение графического файла
`imfinfo('файл',...)`	информация о графическом файле
`image(матрица,...)`	графический образ матрицы
`colormap(матрица(m x 3))`	задание палитры
`imagesc(матрица,...)`	отображает матрицу, каждый элемент матрицы соответствует прямоугольному участку образа.
`surface(матрица,...)`	отобразить матрицу как поверхность над прямоугольной областью
`surf(матрица,...)`	построение графика поверхности
`mesh(матрица,...)`	построения каркасного изображения

Несколько замечаний о работе в MATLAB. Чтобы в следующем сеансе запуска среды MatLab продолжить работу с теми же переменными, что и в текущем, надо сохранить рабочее пространство (workspace). Чтобы сохранить все рабочее пространство целиком, достаточно щелкнуть мышью в любом месте окна рабочего пространства, а затем выбрать в появившемся меню строку Save Workspace As (сохранить рабочее пространство под именем), в которой можно набрать имя и указать папку, в которой сохранить этот файл. Для завершения команды следует нажать кнопку Save. Файлы сеанса сохраняются с расширением *.mat. В старых версиях Matlab в окне Workspace для сохранения рабочего пространства имеется кнопка.

Для загрузки всего сохраненного рабочего пространства или части переменных необходимо щелкнуть мышью на значке папки на панели инструментов окна рабочего пространства. Это откроет папку с MAT-файлами, из которых можно выбрать интересующий файл. Если щелкнуть по выбранному файлу или нажать кнопку Open, то это действие откроет содержимое файла, которое будет загружено рабочее пространство MATLAB.

Все указанные действия можно также совершить из командной строки, если набрать команды `save` и `load`, после которых поместить имена соответствующих файлов с их путями. Если путь не будет указан, то файлы сохраняются в текущем каталоге MatLab. Например, команда `save test` сохранит рабочее пространство в файл `test.mat` текущего каталога.

2. Преобразование яркости и пространственная фильтрация

2.1 Преобразование яркости.

Как мы говорили в начале, результатом дискретизации и квантования изображения является матрица чисел. В результате цифровое изображение состоит из конечного числа элементов, каждый из которых расположен в конкретном месте и имеет определенное значение. Эти элементы принято называть пикселями.

Предположим, что изображение $f(x,y)$ после дискретизации представлено в виде матрицы, у которой имеется `M` строк и `N` столбцов. Тогда говорят, что изображение имеет размер `M x N`. В литературе по обработке изображений за начало координат принимается верхний левый угол изображения, координатами которого служит пара (0, 0). Следующая точка в первой строке изображения имеет координаты (0,1). Это, однако, не обозначает настоящие физические координаты при регистрации соответствующего изображения. Фактически точки изображения мы задаем их индексами (i, j), например, i – номер строки с отсчетом от нуля, а j – номер столбца. Однако мы будем придерживаться обозначения (x, y).

Обозначения массивов, используемые в пакете IPT MatLab, отличается от описанного выше тем, что левый верхний пиксель имеет координаты (1,1)

а переменные номера строк и столбцов изменяются от 1 до M и от 1 до N. Т.о. цифровое изображение имеет естественное представление в виде матрицы

$$\mathtt{f} = \begin{bmatrix} \mathtt{f}(1,1) & \mathtt{f}(1,2) & \cdots & \mathtt{f}(1,N) \\ \mathtt{f}(2,1) & \mathtt{f}(2,2) & \cdots & \mathtt{f}(2,N) \\ \vdots & \vdots & & \vdots \\ \mathtt{f}(M,1) & \mathtt{f}(M,2) & \cdots & \mathtt{f}(M,N) \end{bmatrix},$$

Кроме того, мы предполагаем, что дискретные уровни яркости расположены с постоянным шагом (т.е. используем равномерное квантование). Интервал значений яркости $(f_{\min}, f_{\max})$, где $f_{\min}, f_{\max}$ – минимальное и максимальное значение матрицы `f`, называют динамическим диапазоном изображения. Если заметная доля пикселей занимает значительную часть всего диапазона уровней серого, то мы говорим, что изображение имеет высокий контраст. Наоборот, изображение с малым динамическим диапазоном обычно выглядит тусклым, размытым и серым. Увеличение динамического диапазона изображения делает его более контрастным.

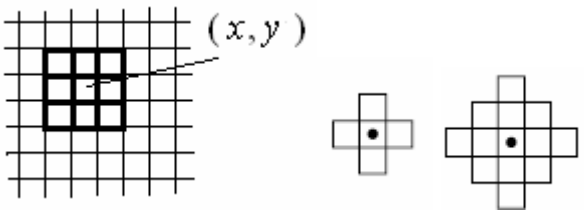
Методы, которые мы будем рассматривать в этой главе, направлены на улучшение изображений. При этом следует иметь в виду, что визуальное оценивание качества изображений является достаточно субъективным.

Процедуры обработки пространственной области изображения оперируют напрямую с пикселями изображения. Методы обработки изображения, используемые в этой главе, можно описать преобразованием

$$g(x,y) = T[f(x,y)] \qquad (1)$$

где $f(x,y)$ – входное изображение, $g(x,y)$ – выходное (обработанное) изображение, а T — некоторый оператор (преобразование) над f, который определен в некоторой окрестности точки (x, y). В ряде случаев оператор T может обрабатывать несколько изображений, например, он может суммировать или усреднять N входных изображений.

Основа этих методов состоит в определении пространственной окрестности вокруг точки (x, y), которая может быть квадратной, прямоугольной или дугой по форме областью с центром в точке (x, y), как показано на рисунке.

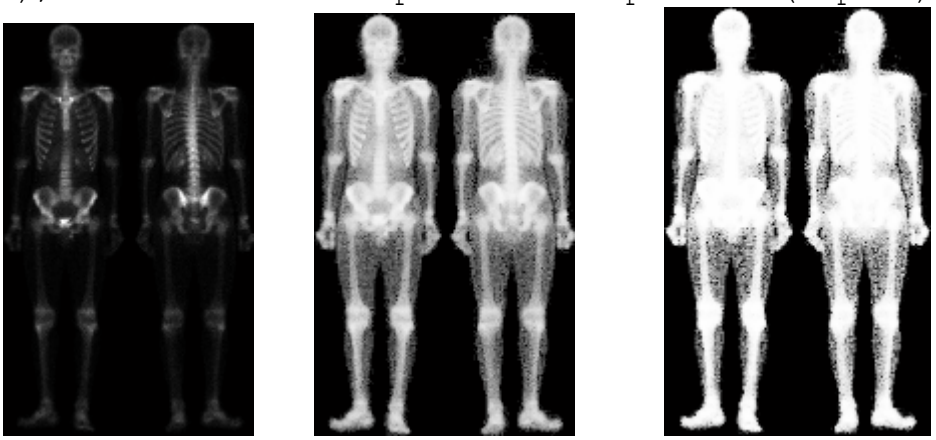


Центр заданной шаблонной подобласти перемещается от пиксела к пикселу, начиная, из верхнего левого угла, и на своем пути накрывает различные окрестности. Преобразование T применяется в каждой точке (x, y), давая в результате выходное (обработанное) значение g для данной точки. В процессе вычислений используются только пикселы внутри заданной окрестности с центром в (x, y).

Простейшая форма преобразования T получается, когда окрестность на рисунке имеет размер `1 x 1` (т.е. состоит из одного пиксела). В этом случае значение g в точке (x, y) зависит только от значения f в этой точке, и T становится функцией преобразования яркости.

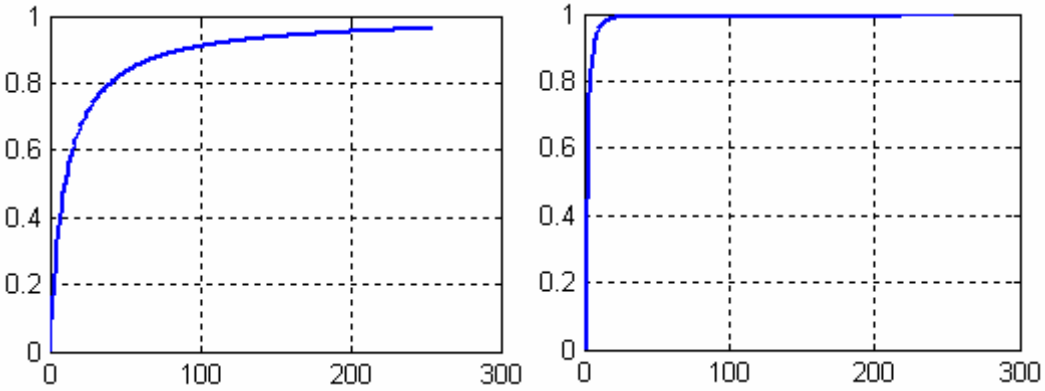
Пример 1. Усиление контрастности. На следующем рисунке слева показано входное изображение *f*. Применяя к нему преобразования $g_1 = \frac{f}{f+10}$ и $g_2 = \frac{f^2}{f^2+5}$, мы получаем изображения, приведенные в центре и справа.

```
f=imread('Bone2.tif');
imshow(f);      % исходное изображение (слева)
g1=double(f)./(double(f)+10);
imshow(g1);     % растяжения контрастности
g2=double(f).^2./(double(f).^2+5);
imshow(g2);     % слишком контрастное изображение (справа)
```



Кривые, используемые для получения среднего и правого изображения и выполняющие преобразование контрастности, имеют следующие графики

```
x=0:1:255; y=x./(x+10);plot(x,y,'LineWidth',2); grid on;
x=0:1:255; y=x.^2./(x.^2+5);plot(x,y,'LineWidth',2); grid on;
```



Откладывая на горизонтальной оси значение яркости точки, которое в нашем примере лежит в диапазоне от 0 до 255, на вертикальной оси находим значение преобразованной яркости, которое для изображения класса `double` лежит в диапазоне от 0 до 1. Для определения максимального значения яркости преобразованного изображения можно использовать команду

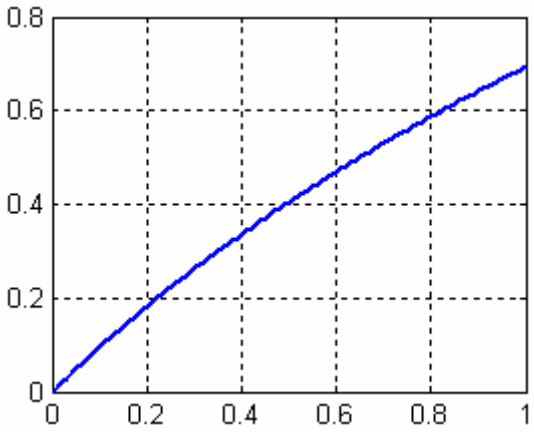
```
max(max(g1))
ans =
    0.9623
```

□

Функции, выполняющие преобразование (1) могут выбираться по-разному. Часто используется логарифмическое преобразование, которое выполняется с помощью выражения

$$g = c \cdot \log\left(1 + double\left(f\right)\right) \qquad (2)$$

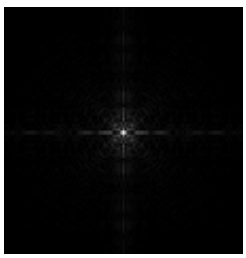
где c — некоторая константа. Форма этой кривой при c=1 и $0 \le f \le 1$ показана на следующем рисунке.

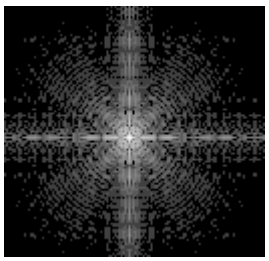


Для реального изображения значения f и g должны быть преобразованы к диапазону, например, 0 – 255.

Пример 2. Использование логарифмического преобразования. Исходное изображение показано слева.

```
f=imread('Spectrum2.tif');
imshow(f);   % изображение слева
```



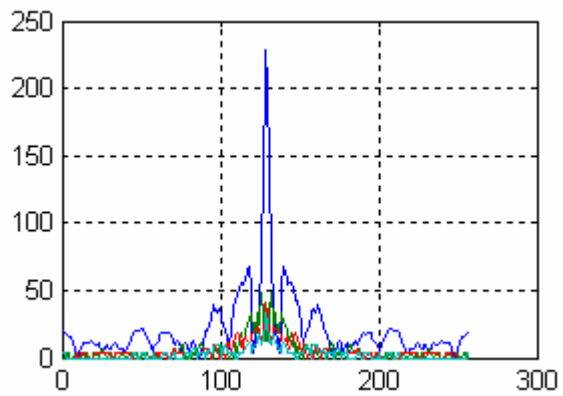


Узнаем размер изображения и найдем максимальное значение в массиве f.

```
size(f)
ans =
   257   257
max(max(f))
ans =
  255
```

Построим его значения по некоторым горизонтальным прямым

```
x=[1:1:257];
plot(x,f(128,:),x,f(120,:),x,f(110,:),x,f(100,:));
```



Как видим, большая часть изображения имеет значительно меньшие значения яркости, чем в точке максимума. Поэтому как сама яркость мала, так и разница в яркостях точек, расположенных вдали от центра мала и практически незаметна. Справа от исходного изображения показан результат применения команды логарифмического преобразования изображения

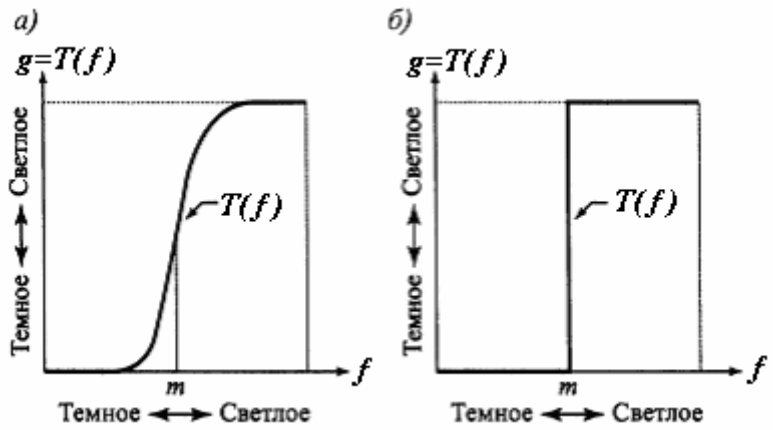
```
g = im2uint8(mat2gray(log(1+double(f))));% натуральный логарифм
imshow(g);
```

Заметим, что здесь мы использовали функции `mat2gray` переводящую элементы матрицы `f` к диапазону [0 1] и функцию `im2uint8`, которая преобразует матрицу-аргумент к диапазону [0 255]. Вот пример последовательного применения этих функций

```
A=[1 2 3; 4 5 6; 7 8 9]
B=mat2gray(A)
C=im2uint8(B)
A =                    B =                              C =
     1     2     3    0         0.1250    0.2500          0   32   64
     4     5     6    0.3750    0.5000    0.6250         96  128  159
     7     8     9    0.7500    0.8750    1.0000        191  223  255
```

□

Функция, показанная на следующем рисунке слева, называется функцией преобразования растяжения контрастности, поскольку она сжимает входные величины, меньшие чем `m`, в более узкий поддиапазон темных уровней на выходном изображении, и, соответственно, величины, большие `m`, — в более узкую полосу ярких уровней. В результате получается изображение с большей контрастностью.



Уравнение такой кривой можно записать в виде

$$g = T(f) = \frac{1}{1 + (m/f)^E} \qquad (3)$$

где f — это яркость входного изображения, g — соответствующая яркость выходного изображения, а параметр E контролирует наклон кривой. Это уравнение реализуется на MATLAB в виде следующей команды

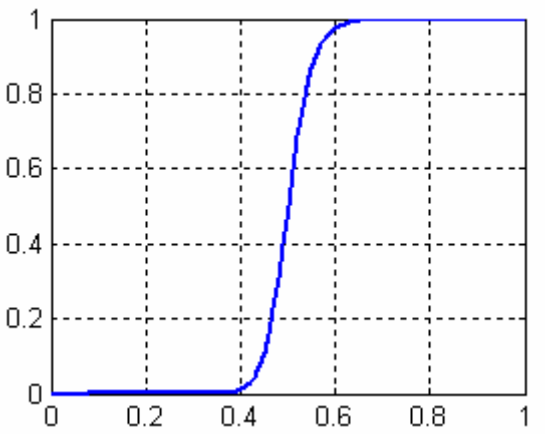
```
g = 1./(1 + (m./(double(f) + eps)).^E)
```

Использование величины `eps` позволяет избежать ошибки переполнения, если в f имеются нулевые значения. Поскольку верхнее предельное значение функции T(f) равно 1, выходные значения масштабированы в диапазоне [0,1].

В предельном случае, показанном на предыдущем рисунке справа, выходом служит двоичное (черно-белое) изображение. Эта предельная функция, называемая пороговой, является простейшим инструментом при сегментации изображений.

Например, кривая, приведенная на следующем рисунке, получена следующими командами

```
f=[0:0.01:1];
g=1./(1+(0.5./(f+eps)).^20);
plot(f,g,'LineWidth',2); grid on;
```



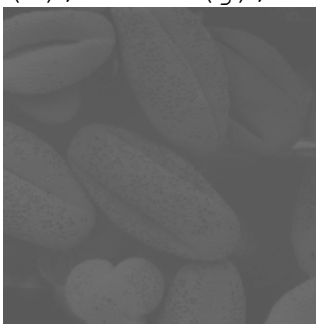
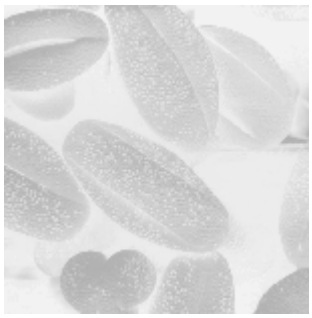
Следующий код, использующий формулу (3), дает тот же результат, что и в примере 1. Фактически в том примере мы использовали другую запись тех же функций преобразования

```
f=imread('Bone2.tif');
imshow(f);      % исходное изображение (слева)
g1=1./(1+(10./(double(f)+eps)).^1);
imshow(g1);     % применение растяжения контрастности
g2=1./(1+(5.0./(double(f)+eps)).^2);
imshow(g2);     % слишком контрастное изображение (справа)
```

Часто удобными являются кусочно-линейные функции преобразования изображений.

Пример 3. На следующем рисунке представлено исходное малоконтрастное 8-битовое изображение - фотография пыльцы, сделанная на электронном микроскопе. Построение негатива с помощью функции `imcomplement` ситуацию улучшает незначительно.

```
f=imread('Pollen2.tif'); imshow(f);
g = imcomplement(f);imshow(g);
```

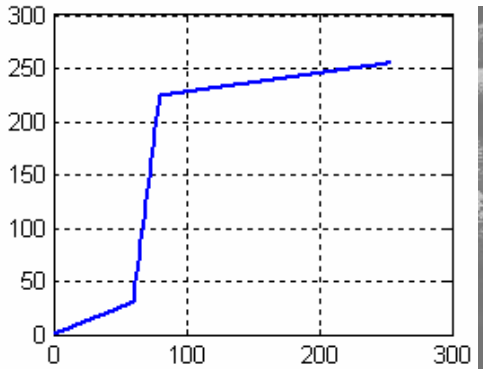



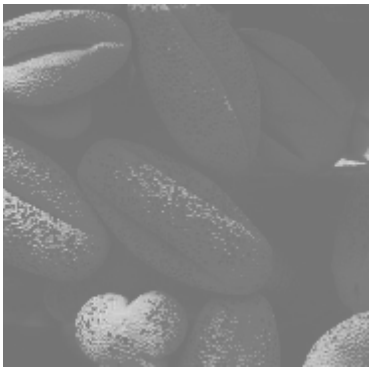
Для обработки изображения с помощью ломанной, имеющей два излома, построим функцию, генерирующую ломаную

```
function z=polyline2(x,pnt)
% уравнение ломаной, с двумя изломами в точках (x1,y1) и (x2,y2)
% точки излома передаются в векторе pnt=[x1 y1 x2 y2].
% Ломаная обязательно проходит через точки (0,0) и (255, 255)
% Точки излома находятся в диапазоне 0<x1<x2<255, 0<y1<y2<255
x1=pnt(1);y1=pnt(2);x2=pnt(3);y2=pnt(4); % две точки излома
z0=y1/x1.*x;
z2=y2+(255-y2)/(255-x2).*(x-x2);
z=0.5.*(z0+z2)+0.5.*(((y2-y1)/(x2-x1)-y1/x1).*abs(x-x1)+...
     ((255-y2)/(255-x2)-(y2-y1)/(x2-x1)).*abs(x-x2));
```

На следующем рисунке показан график ломаной, используемой для повышения контраста изображения, и результат ее использования.

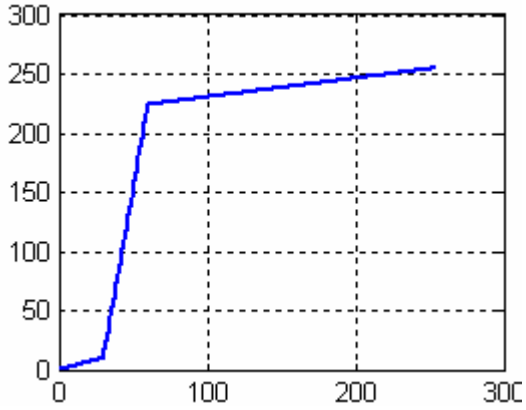
```
x=0:1:255; pnt=[60 30 80 224];
plot(x,polyline2(x,pnt),'LineWidth',2); grid on;
f=imread('Pollen2.tif');g=polyline2(f,pnt); imshow(g);
```

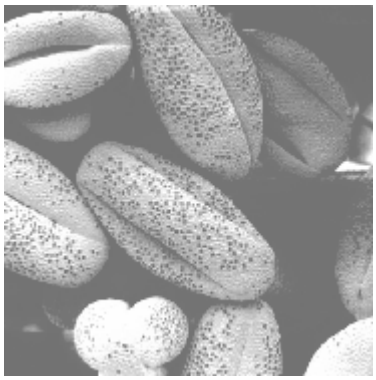




Еще лучший результат получается для ломаной, сдвинутой ближе к темным тонам.

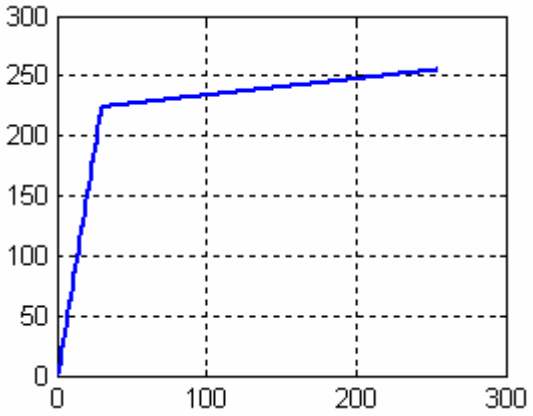
```
x=0:1:255; pnt=[30 10 60 224];
plot(x,polyline2(x,pnt),'LineWidth',2); grid on;
f=imread('Pollen2.tif'); g=polyline2(f,pnt); imshow(g);
```

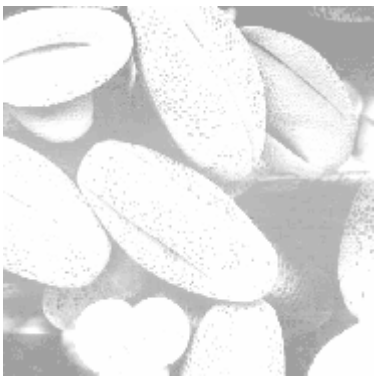




Однако слишком сильное приближение к оттенкам черного дает неудовлетворительный результат.

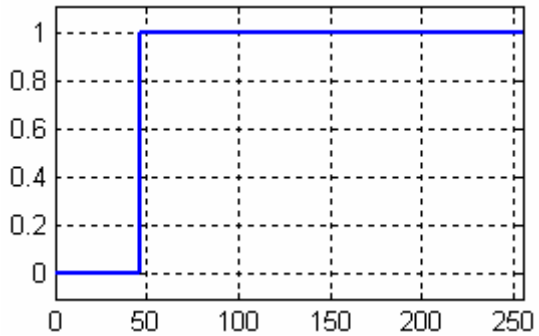
```
x=0:1:255; pnt=[1 1 30 224];
plot(x,polyline2(x,pnt),'LineWidth',2); grid on;
f=imread('Pollen2.tif');g=polyline2(f,pnt); imshow(g)
```





Результат порогового преобразования показан ниже

```
f=imread('Pollen2.tif'); max(max(f))
ans =
   83
x=0:255; plot(x,double(x)>45, 'LineWidth',2); grid on;
axis([ 0 255 -0.1 1.1]);  set(gcf,'Color',[1 1 1]);
g1=double(f).*(double(f)>45);  imshow(g1);
```

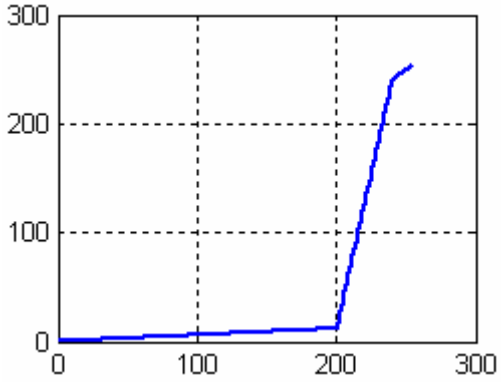


□

Пример 4. Вот еще пример применения кусочно – линейного преобразования, использующий нашу функцию `polyline2`.

```
f=imread('Aero2.tif'); imshow(f);
```

```
c=220; d=20;   % подбираем форму ломаной
x=0:1:255; pnt=[c-d 12 c+d 240];
plot(x,polyline2(x,pnt),'LineWidth',2); grid on;
f=imread('Aero2.tif'); g=polyline2(f,pnt); imshow(g);
```



Как видим, контраст изображения увеличился.

□

Подбирать преобразующую яркость кривую иногда бывает сложно. Для облегчения такого подбора пакете IPT MATLAB есть достаточно универсальные функции. Рассмотрим одну из них.

Функция `imadjust` является базовым инструментом пакета IPT при преобразованиях яркости полутоновых изображений. Она имеет следующий синтаксис:

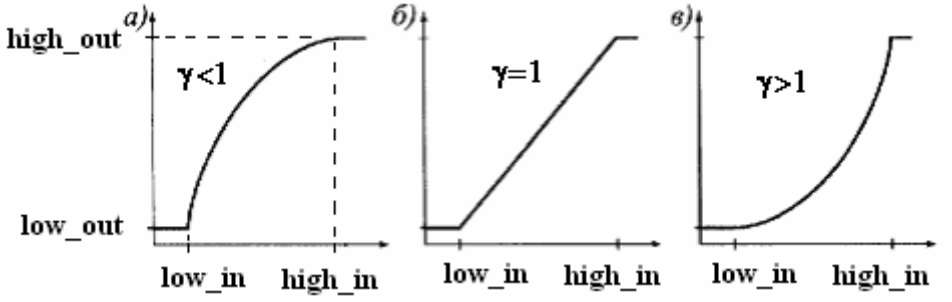
```
g=imadjust(f,[low_in,high_in],[low_out,high_out],gamma)
```

Она выполняет преобразование яркости $g = T(f)$ исходного изображения f в новое изображение g, при котором значения яркости в интервале `[low_in, high_in]` переходят в значения интервала `[low_out, high_out]`, а значения, меньшие порога `low_in` или большие порога `high_in`, обрезаются. Т. е. все, что меньше `low_in`, отображается в `low_out`, а все, что больше `high_in`, отображаются в `high_out`.

Входное изображение может иметь класс `uint8`, `uint16` или `double`, и класс выходного изображения совпадает с классом входного. Это значит, что элементы матрицы изображения (монохромного) являются беззнаковыми целыми числами, изменяющимися в диапазоне 0 – 255 (`uint8`); беззнаковыми целыми числами, изменяющимися в диапазоне 0 – 65535 (`uint16`); или вещественными числами (`double`) обычно изменяющимися в диапазоне 0 – 1.

Все входные параметры функции `imadjust`, за исключением *f*, должны быть вещественными числами в интервале от 0 до 1, независимо от класса *f*. Если *f* принадлежит классу `uint8`, функция `imadjust` умножает эти параметры на 255 для задания истинных величин, которые будут использоваться; если `f` — класса `uintl6`, то все умножается на 65535. Если вместо векторов `[low_in, high_in]` или `[low_out, high_out]` поставить пустой вектор ([ ]), то будут использоваться величины по умолчанию, равные [0 1]. Если `high_out` меньше, чем `low_out`, то выходные яркости симметрично переворачиваются (получается негатив).

Параметр `gamma` служит для задания формы кривой, отображающей яркость *f* в яркость *g*. Если *gamma* меньше 1, то яркость отображения смещается вверх в сторону более ярких значений, как показано ниже (*а*). Если `gamma` больше 1, то яркость отображения смещается вниз в сторону менее ярких значений (*в*). Если параметр `gamma` опущен, то его значение по умолчанию равно 1 (линейное отображение *б*). Возможные графики преобразующих кривых приведены на следующем рисунке.



Пример 5.

```
f=imread('Moon2.tif');
size(f)
     ans =   540   466
imshow(f);          % изображение слева
g1 = imadjust(f, [0 1], [1 0]);
imshow(g1);         % изображение в центре (негатив)
```

Процедура `imadjust(f,[0 1],[1 0])` является цифровым эквивалентом получения фотонегатива и весьма полезна для усиления белых или серых участков, окруженных большими, преимущественно темными, областями.

Негативное изображение можно легко построить также с помощью функции `imcomplement` из пакета IPT:

```
g2 = imcomplement(f);
imshow(g2);
```

Результатом будет негативное изображение луны (такое же как на предыдущем рисунке в центре). На предыдущем рисунке справа приведены результаты выполнения команд

```
g3 = imadjust(f, [0.5 0.75], [1 0]);
imshow(g3);          % изображение справа
```

которые растягивает шкалу градации между 0.5 и 0.75 на весь интервал [0,1].

Наконец, по команде

```
g4 = imadjust(f, [ ], [ ] , 0.3) ;
imshow(g4);
```

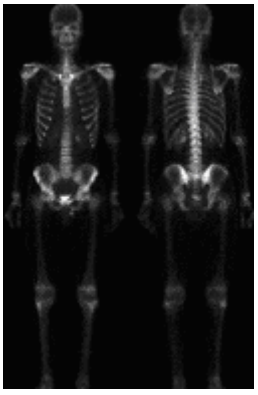
получаем изображение с большими серыми тонами (следующий рисунок слева). Можно также построить негатив с меньшими оттенками серого (рисунок справа)

```
g5 = imadjust(f, [ ], [1 0 ] , 0.7) ;
imshow(g5);
```

При этом производится растяжение нижнего и сжатие верхнего участка градационной шкалы.

Вот еще один пример построения негативного изображения (см. пример 1)

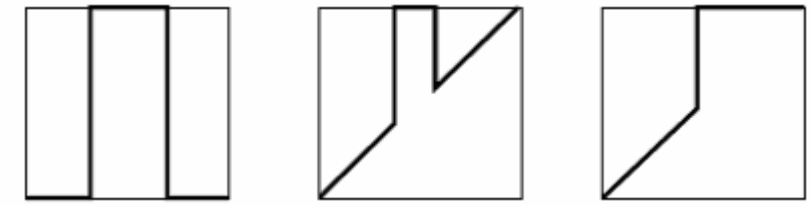
```
f=imread('Bone2.tif');
imshow(f);     % исходное изображение (слева)
g = imcomplement(f);
imshow(g);     % негативное изображение
```



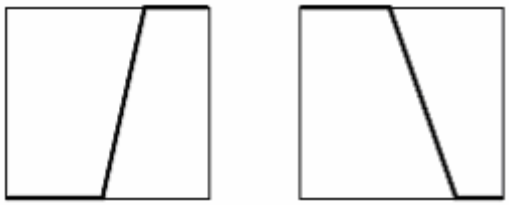
Функции поэлементных преобразований яркости могут иметь различную форму. Перечислим некоторые другие виды таких преобразований .

Обобщением порогового преобразования является преобразование яркости среза (след рисунок слева). Оно позволяет выделить определенный интервал диапазона яркостей входного изображения. Перемещая «рабочий» интервал по шкале и меняя его ширину, можно производить визуальный

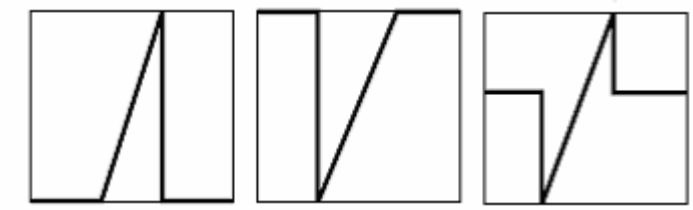
анализ объектов, различающихся по яркости. Детали, не попавшие в выбранный интервал, будут подавлены. На следующем рисунке в середине приведен график функции преобразования среза с сохранением фона. В этом случае изображение в целом сохраняется, но на нем ”высвечиваются” участки, попавшие в заданный интервал яркостей. Если этот интервал примыкает к границе шкалы яркости, то получаем преобразование называемое неполной пороговой обработкой (график справа)



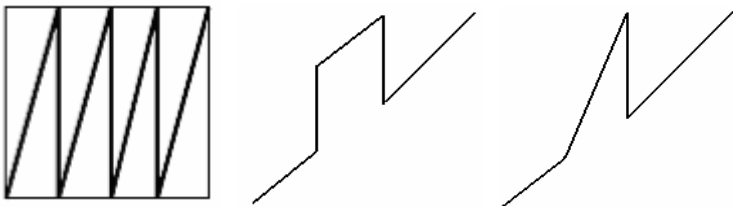
Контрастное масштабирование, график которого показан на следующем рисунке слева, нами уже рассматривался. Здесь «рабочий» интервал яркостей растягивается на весь диапазон допустимых значений. Контрастное масштабирование, соответствующее графику справа, обращает функцию яркости, т.е. дает негатив.



Контрастное масштабирование, графики которого показаны на следующем рисунке, выделяет «рабочий» диапазон яркостей на черном фоне (след рисунок слева), на белом фоне (рисунок в середине), на сером фоне (график справа).



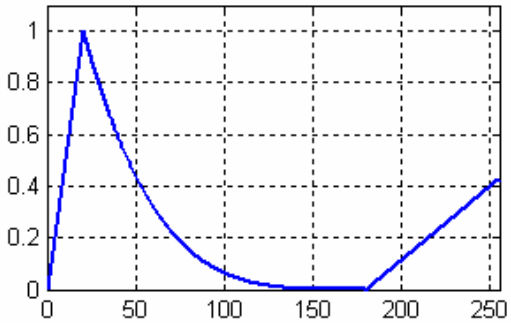
Пилообразное контрастное масштабирование (след. рисунок слева) удобно применять для изображений, состоящих из нескольких областей с медленно меняющимся значением яркости. Такое преобразование почти не разрушает целостности восприятия изображения, но резко увеличивает контрастность плохо различимых мелких деталей. На следующем рисунке в середине и справа приведены еще два графика возможных полезных преобразований.



Можно создавать кусочные преобразования и более изощренной формы. Приведем один такой пример с кусочно заданной функцией преобразования

Пример 6. (ex7.m)

```
% Функция преобразования
z=@(x) x/20.*(x<20)+ ((x-180).^4/160^4).*(x>=20 & x<=180) +
(x-180)/175.*(x>180);
x=0:1:255;
plot(x,z(x),'LineWidth',2); grid on; set(gcf,'Color',[1 1 1]);
axis([0 255 0 1.1]);
% исходное изображение
f=imread('Phobos2.tif');  % чтение исходного изображение
figure,
imshow(f);
g=z(double(f))*255;
figure, imshow(g);      % результирующее изображение
```




В этом примере нам удалось выделить в темной части изображения некоторые детали, которые не были видны на исходном изображении. Обратите внимание на способ создания кусочной функции в этом сценарии. Его особенностью является использование логических выражений типа `(x<20)` или `(x>=20 & x<=180)`, которые для каждого элемента вектора или матрицы `x` принимают значение ноль или единица.

□

Выбор подходящего преобразования не всегда является простой задачей. Иногда в этом нам может помочь гистограмма изображения.

Гистограммой цифрового изображения называется дискретная функция

$$h(r_k) = n_k$$

где r_k — это k - й уровень яркости (из интервала квантованных яркостей [0,G]), а n_k — число пикселов изображения, уровень яркости которых равен r_k. Значение G равно 255 для изображений класса uint8, 65535 — для класса uint16 и 1.0 — для класса double.

Основной функцией пакета для обращения с гистограммами служит функция imhist со следующим синтаксисом:

```
h = imhist(f, b),
```

где f — это входное изображение, h - его гистограмма $h(r_k)$ (фактически вектор значений), и b — число корзин, использованных при формировании гистограммы (если аргумент b отсутствует, то по умолчанию принимается b = 256). Корзиной называется подразделение шкалы яркости. Например, если при работе с изображениями класса uint8 переменная b = 2, то шкала яркости делится на две подобласти (корзины): от 0 до 127 и от 128 до 255. Итоговая гистограмма будет иметь два значения: h(1), равное числу пикселов

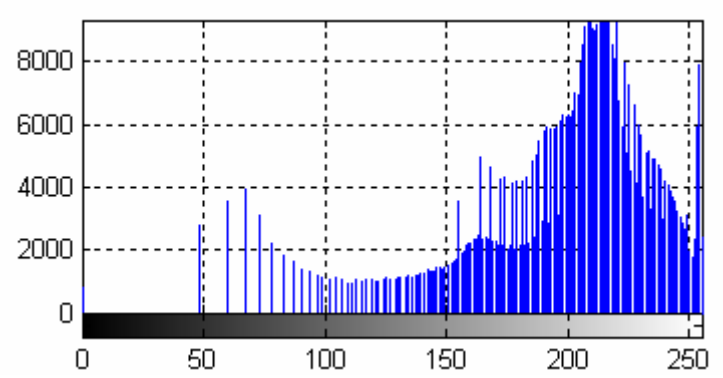
изображения, величины которых находятся в интервале [0,127], и h(2), которое равно числу пикселов со значениями в интервале [128, 255]. Чтобы получить нормированную гистограмму, надо выполнить деление

```
p = imhist(f, b)/numel(f),
```

где `numel(f)` это число элементов массива `f`, т.е. число пикселов изображения.

Пример 7. Нахождение гистограмм и построение их графиков.

```
f=imread('Aero2.tif'); numel(f)
ans =
      586752
imhist(f);  % гистограмма по умолчанию
grid on;
```

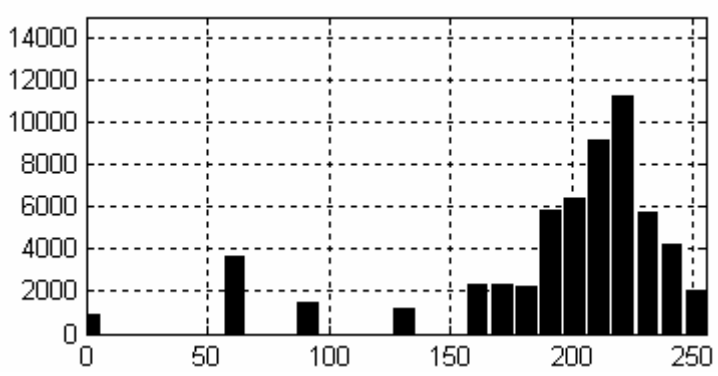


□

Однако имеется много других способов построения гистограммы.

Гистограмму можно построить с помощью столбчатых диаграмм. Для этого служит функция `bar(horz,v,width)`, где `v` — это вектор-строка, график которой надо построить, `horz` — вектор того же размера, что и вектор `v`, состоящий из значений по горизонтальной шкале, и `width` — число между 0 и 1. Если параметр `horz` опущен, то горизонтальная ось делится с шагом 1 от 0 до `length(v)`. Когда `width` равно 1, соседние столбики соприкасаются, когда `width` равно 0, столбики являются вертикальными линиями, как на предыдущем рисунке. По умолчанию `width` равно 0.8. Следующие команды приводят к построению столбчатой диаграммы, на которой горизонтальная ось делится на группы по 10 уровней:

```
h=imhist(f); h1 = h(1:10:256); horz = 1:10:256;
bar (horz, h1); grid on;      % построение гистограммы
axis([0 255 0 15000]);
set(gca, 'xtick', 0:50:255)
set(gca, 'ytick', 0:2000:15000)
set(gcf,'Color',[1 1 1]);     % белый фон графического окна
```



Функция `axis` имеет синтаксис

```
axis([horzmin horzmax vertmin vertmax]),
```

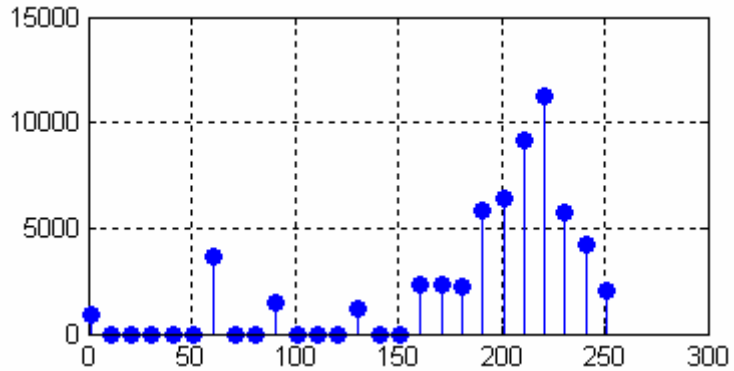
где устанавливаются минимальные и максимальные значения горизонтальных и вертикальных координат. В последних двух командах `gca` означает фразу «get current axis» (использовать текущие оси, т. е. оси только что построенного графика), а `xtick` и `ytick` устанавливают метки на горизонтальной и вертикальной оси через заданные интервалы.

Стеблевая диаграмма строится аналогично столбчатой диаграмме. Ее синтаксис имеет вид

```
stem(horz,v,'color_linestyle_marker', 'fill'),
```

где `v` — это вектор-столбец, график которого необходимо построить, а переменная `horz` имеет тот же смысл, что и в функции `bar`. Аргумент `color_linestyle_marker` состоит из комбинации символов, управляющих стилем диаграммы. Например, команда `stem(v,'r s')` построит стеблевую диаграмму, на которой вертикальные стебли будут проведены красными пунктирными линиями, а головки стеблей будут квадратными. Если присутствует переменная `fill`, а переменная `marker` обозначает круг, квадрат или ромб, то эти символы внутри закрашиваются в цвет, предписанный переменной `color`. По умолчанию цвет считается черным, линии стеблей сплошными, а головками служат круги. Стеблевая диаграмма на следующем рисунке получена с помощью команд

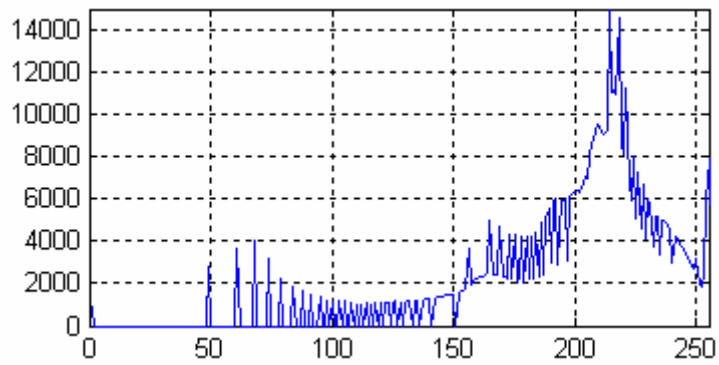
```
h=imhist(f); h1 = h(1:10:256);
horz = 1:10:256;
stem(horz,h1,'o','fill');  % 'o' - кружочки, 'fill' - закрашены
set(gcf,'Color',[1 1 1]); grid on;
```



Функцию `plot` строит график по точкам, соединяя их отрезками прямых линий. Она имеет синтаксис

`plot(horz,v,color_linestyle_marker')` ,
где аргументы имеют тот же смысл, что и у функции `stem`.

```
h=imhist(f); plot(h);
axis([0 255 0 15000]); grid on;
```



Теперь мы знаем, что наше изображение `f` имеет очень мало пикселей с яркостью в диапазоне 0 – 50. Поэтому можно применить преобразование яркости, с подавлением этих пикселей (изображение в центре). Оставляя динамический диапазон яркости 150 – 255 мы получим еще более контрастное изображение (изображение справа).

```
f=imread('Aero2.tif'); imshow(f); % изображение слева
g1=imadjust(f,[0.2 1],[0 1],1);imshow(g1); % изображение в центре
g2=imadjust(f,[0.6 1],[0 1],1);imshow(g2); % изображение справа
```



Таким образом, если на гистограмме/графике есть сгруппированные области, то для повышения контрастности изображения надо растянуть диапазон яркости, соответствующий этой области.

Пусть в исходном изображении имеется `L` градаций яркости $r_1 < r_2 < \ldots < r_L$, которые встречаются в соответственно $n_1, n_2, \ldots, n_L$ раз (т.е. значение яркости r_i имеют n_i пикселей) и, положим $n = \sum_{j=1}^{L} n_j$. Преобразуем изображение следующим образом – заменим градации r_k на величины

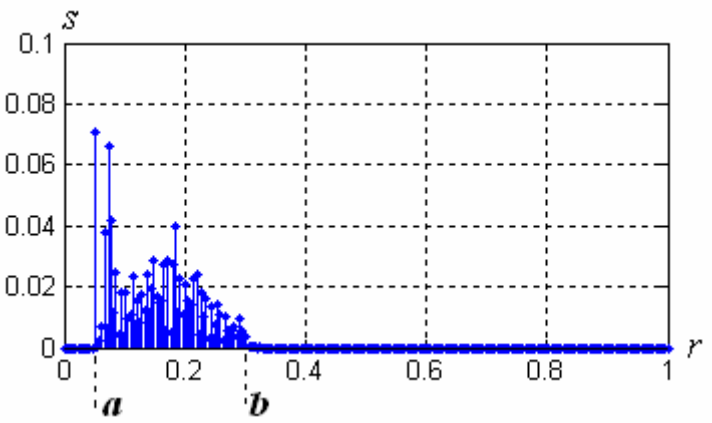
$$s_k = T(r_k) = \sum_{j=1}^{k} \frac{n_j}{n} \quad (k = 1,\ldots,L). \qquad (4)$$

т.е. s_k будут величинами яркости выходных пикселей, имевших в исходном изображении яркость r_k. При этом очевидно, что все $s_k \le 1$, поскольку в

сумме (4) стоят только положительные слагаемые и самое большое значение может иметь значение $s_L = \sum_{j=1}^{L} \frac{n_j}{n} = \frac{1}{n} \sum_{j=1}^{L} n_j = 1$.

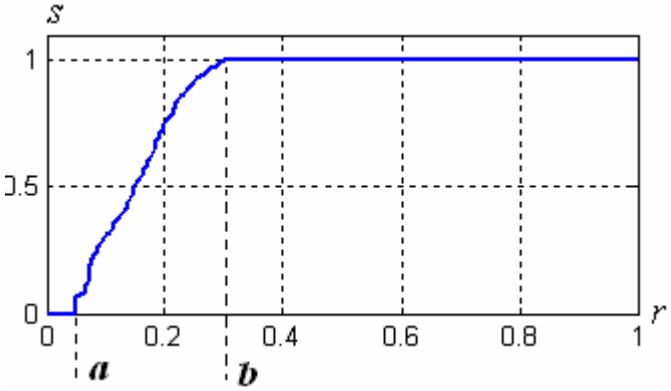
Результат преобразования (4), называемого эквализацией изображения, состоит в увеличении динамического диапазона уровней яркости, что обычно означает большую контрастность выходного изображения. Действительно, пусть изображение имеет отчетливо выделенную область яркостей, например, такую как представляет следующая нормированная гистограмма

```
f=imread('Pollen2.tif');
hnorm = imhist(f)./numel(f);
x = linspace(0, 1, 256);  % вектор 256 значений от 0 до 1
stem(x,hnorm,'.'); axis([0 1 0 0.1]);
set(gcf,'Color',[1 1 1]); grid on;
```



Тогда преобразование (4) даст функцию $s = T(r)$, которая будет нулем (или очень мала) на участке $r < a$, монотонно возрастающей на отрезке $[a\ b]$ и константой (или почти не возрастающей) на участке $r > b$. При этом растяжение области яркостей будет как раз в том месте, где сосредоточены основные градации яркости изображения. Функция преобразования будет иметь следующий вид

```
hnorm = imhist(f)./numel(f);  % нормировка к диапазону 0 - 1
sdf = cumsum(hnorm);
x = linspace(0, 1, 256); % вектор 256 значений от 0 до 1
plot(x,sdf,'LineWidth',2); set(gcf,'Color',[1 1 1]);
grid on; axis([0 1 0 1.1]);
```



Т.о. эта функция преобразует узкий входной диапазон яркостей $a \le r \le b$ на всю шкалу яркости $0 \le s \le 1$ выходного изображения.

Здесь мы использовали функцию `cumsum` накапливающего суммирования. Если A – вектор длины N, то `B=cumsum(A)` – вектор элементов B_k вычисляемых по формуле $B_k = \sum_{i=1}^{k} A_i \ \ (k = 1,2,...,N)$. Если A — многомерный массив, то `B=cumsum(A,dim)` — накапливаемая сумма элементов вдоль размерности `dim`.

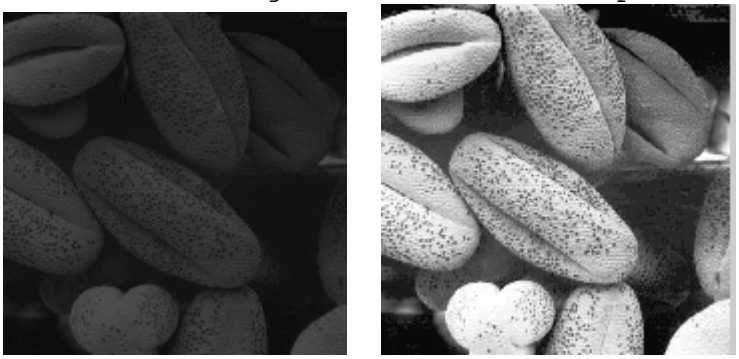
Сама эквализация (т.е. преобразование изображения по формуле (4)) реализована в пакете IPT функцией `histeq`, которая имеет синтаксис

```
g = histeq(f,nlev),
```

где `f` — это входное изображение, а `nlev` — число уровней интенсивности, установленное для выходного изображения. Если `nlev` равно L (общему числу возможных уровней входного изображения), то `histeq` просто реализует функцию преобразования $T(r_k)$. Если число `nlev` меньше, чем L, то `histeq` стремится перераспределить уровни так, чтобы они приближали плоскую гистограмму. В отличие от `imhist`, значением `nlev` в `histeq` по умолчанию является 64. Обычно мы будем использовать максимально возможное число уровней (например, 256) для `nlev`, поскольку это дает истинную реализацию описанного выше метода гистограммной эквализации.

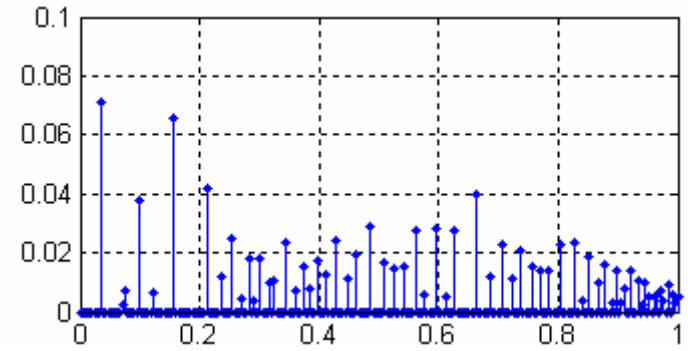
Пример 8. Рассмотрим еще раз изображение пыльцы из файла `Pollen2.tif` и применим к нему преобразование эквализации

```
f=imread('Pollen2.tif'); imshow(f);     % изображение слева
g = histeq(f, 256); imshow(g);          % изображение справа
```



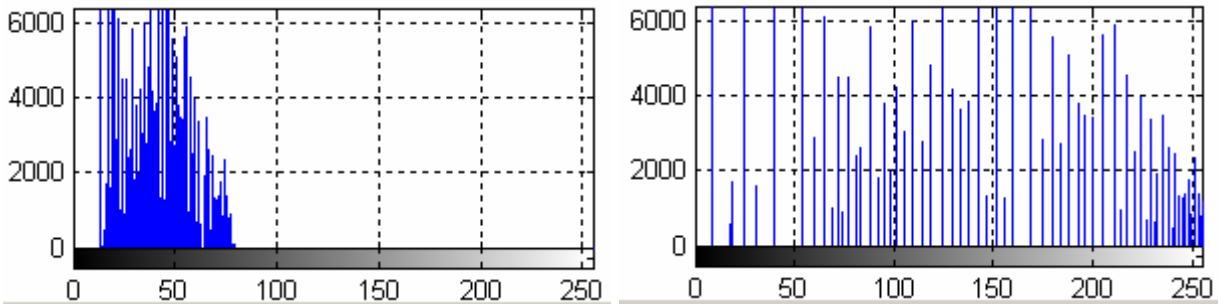
Нормированная гистограмма исходного изображения и соответствующее ему преобразование эквализации показаны нами выше. Нормированная гистограмма преобразованного изображения имеет вид

```
f=imread('Pollen2.tif'); g = histeq(f, 256);
hnorm = imhist(g)./numel(g); x = linspace(0, 1, 256);
stem(x,hnorm,'.'); axis([0 1 0 0.1]); grid on;
```



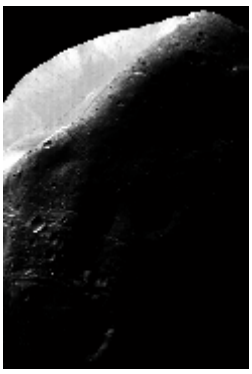
Исходное изображение является весьма темным и у него имеется очень низкий динамический диапазон. Его гистограмма выявляет темную природу изображения, поскольку она (гистограмма) целиком расположена в темном участке спектра. Также очевиден низкий динамический диапазон изображения, поскольку «ширина» гистограммы весьма мала по сравнению с шириной всего диапазона серых полутонов. Улучшение средней яркости и контрастности преобразованного изображения очевидно. То же можно заключить, изучив гистограмму преобразованного изображения. Повышение контрастности вызвано существенным расширением динамического диапазона на всю шкалу яркости. Повышение общей яркости изображения связано с тем, что средний уровень яркости на гистограмме эквализованного изображения стал выше, чем на исходном изображении. Это можно увидеть также построив ненормированные гистограммы исходного и преобразованного изображений

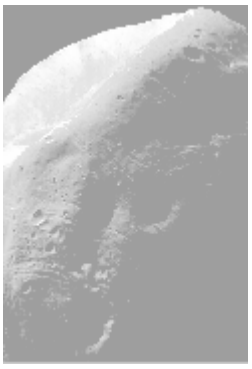
```
figure, imhist(f); grid on;
figure, imhist(g) ; grid on;
```



Гистограммная эквализация совершается преобразованием, которое зависит от гистограммы исходного изображения. Т.е. функция преобразования не будет меняться, пока не изменится само изображение. Как уже отмечалось, гистограммная эквализация улучшает изображение путем расширения диапазона его уровней яркости. Однако такая процедура не всегда приводит к удовлетворительному результату. Поэтому в конкретных приложениях полезно уметь задавать форму гистограммы, которую желательно иметь для обработанного изображения. Метод построения обработанного изображения с заданной гистограммой называется ***гистограммной подгонкой***. Так для следующего изображения гистограммная эквализация не дает существенного улучшения.

```
f=imread('Phobos2.tif'); imshow(f);      % Фобос (слева)
f1=histeq(f, 256); imshow(f1);           % эквализация (справа)
```





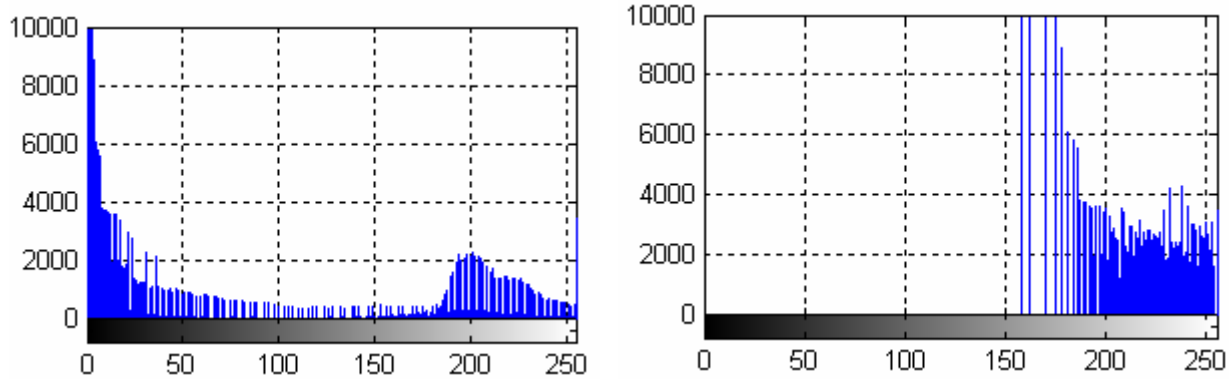
Для построения изображения с заданной гистограммой используется функция `histeq` со следующим синтаксисом

```
g = histeq(f, hspec),
```

где f — это входное изображение, `hspec` — заданная гистограмма (вектор-строка), а `g` — выходное изображение, гистограмма которого близка к заданной гистограмме `hspec`. Вектор `hspec` должен состоять из целых чисел, соответствующих одинаковым разбиениям (корзинам) диапазона уровней. Гистограмма изображения `g`, построенного функцией `histeq`, лучше приближается к `hspec`, когда `length(hspec)` существенно меньше числа уровней яркости исходного изображения `f`.

Пример 9. Гистограммная подгонка. Посмотрим на гистограмму последнего изображения

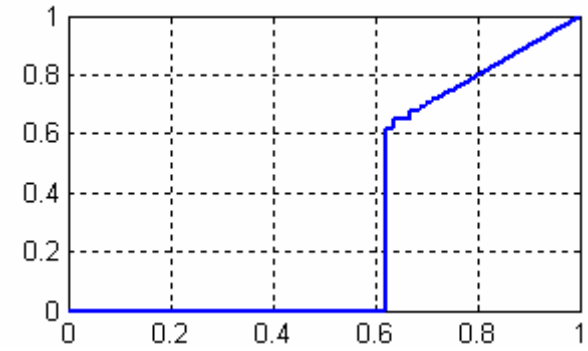
```
imhist(f); set(gcf,'Color',[1 1 1]);
axis([0 255 0 10000]); grid on; % рисунок слева
imhist(f1); % гистограмма эквализованного изображения (справа)
set(gcf,'Color',[1 1 1]);axis([0 255 0 10000]); grid on;
```



На исходном изображении доминируют темные области, что соответствует большой концентрации темных пикселов. Казалось бы, что гистограммная эквализация способна улучшить качество этого изображения, поскольку детали темных областей стали бы более заметными. Однако существенного улучшения изображения не произошло. Причину этого можно объяснить, анализируя гистограмму эквализованного изображения, приведенного на предыдущем рисунке справа. Здесь видно, что уровни яркости были сдвинуты в верхнюю половину градационной шкалы, что привело лишь к общему осветлению изображения, которое теперь выглядит как бы «вылинявшим». Причиной такого сдвига является большая концентрация

значений темных пикселов вблизи нулевого уровня на исходной гистограмме. Оказывается, что функция преобразования, построенная по этой гистограмме, имеет очень крутой подъем.

```
hnorm = imhist(f1)./numel(f1);
sdf = cumsum(hnorm);
x = linspace(0, 1, 256);
plot(x,sdf,'LineWidth',2);
set(gcf,'Color',[1 1 1]); grid on;
```



Одним из возможных средств решения такой проблемы может служить гистограммная подгонка с заданной гистограммой, которая имела бы меньшую концентрацию компонент в нижней части градационной шкалы, но при этом сохраняла бы общую форму гистограммы исходного изображения. Заметим, что исходная гистограмма, в целом, является бимодальной. Это значит, что имеется одна большая группа значений яркости (мода) возле нулевого значения, а другая, меньшая группа (мода), расположена в верхней области градационной шкалы.

Поскольку проблема, возникшая при гистограммной эквализации в этом примере, связана прежде всего с большой концентрацией уровней яркости пикселов возле нулевого значения, разумным решением будет модификация гистограммы изображения, чтобы у нее отсутствовало это явление. Для этого напишем следующий сценарий. Он строит гистограмму исходного изображения (оно приведено нами ранее) и модифицирует его гистограмму,

```
% исходное изображение
f=imread('Phobos2.tif');     % чтение исходного изображения
h0=imhist(f); x = 1:1:256;
bar(x, h0, 0); grid on;      % гистограмма исходного изображения
axis([0 255 0 20000]); set(gcf,'Color',[1 1 1]);

% Построение ломаной, моделирующей форму желаемой гистограммы
z=@(x) 0.5.*(70000/5.*abs(x)+(-65000/15-70000/5).*abs(x-5)+...
        +(-5000/160+65000/15).*abs(x-20)+...
        +(5000/25+5000/160).*abs(x-180)+...
        +(-5000/50-5000/25).*abs(x-205)+(0+5000/50).*abs(x-255));
x=0:0.1:255;
figure,plot(x,z(x),'LineWidth',2); grid on; set(gcf,'Color',[1 1 1]);
axis([0 255 0 75000]);

% моделируем желаемую гистограмму
h01=z(x);
h1=h01./sum(h01).*numel(f);% корректируем гистограмму, чтобы sum(h1)=numel(f)
figure,bar(x, h1, 0); grid on;
```

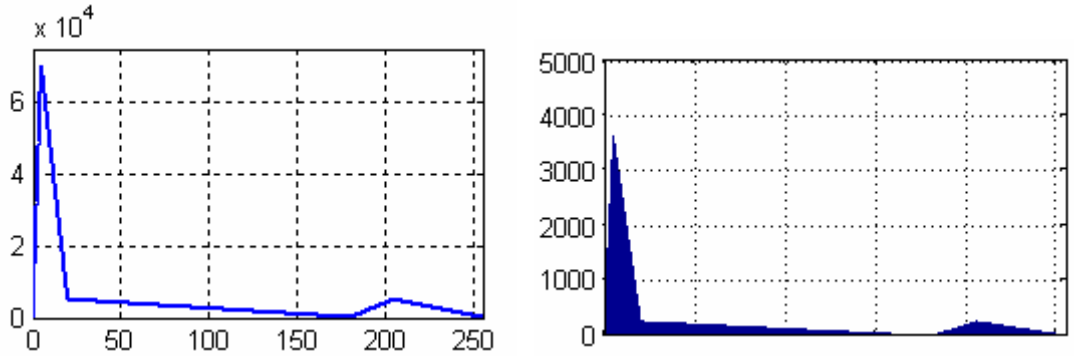
```
axis([0 255 0 5000]); set(gcf,'Color',[1 1 1]);

figure,g = histeq(f, h1); % генерируем изображение с желаемой гистограммой h1
imshow(g);
figure,imhist(g);          %гистограмма изображения, которое получилось

g1=histeq(g, 256); imshow(g1);  % эквализация подправленного изображения
figure,imhist(g1); %гистограмма изображения, которое получилось после
эквализации
% функция преобразования
hnorm = imhist(g1)./numel(g1);
sdf = cumsum(hnorm);
x = linspace(0, 1, 256);
plot(x,sdf,'LineWidth',2);
set(gcf,'Color',[1 1 1]); grid on;
```

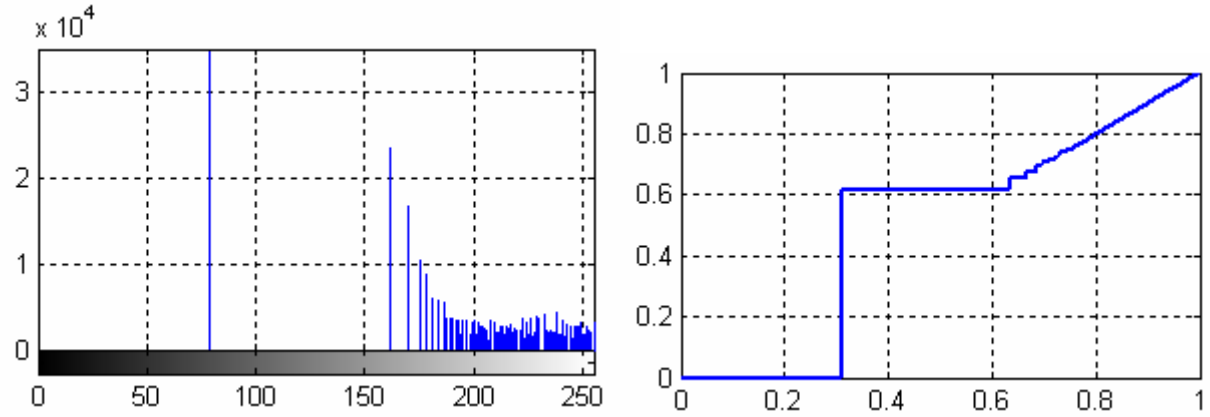
Заметим, что в сценарии мы использовали троеточие для продолжения длинной строки на следующую строку. При этом повторение знака арифметической операции в начале следующей строки необязательно. Сценарий открывает 7 графических окон. Перед повторным запуском все графические окна можно закрыть командой `close all`.

Гистограмма исходного изображения нами приведена выше. Вот график ломаной и гистограммы, построенной по ней. Вертикальные значения новой гистограммы по сравнению с графиком ломаной мы скорректировали, поскольку количество пикселей, которое она будет представлять, должно равняться количеству пикселей исходного изображения.



Изображение `g`, построенное по правой гистограмме командой `g=histeq(f,h1)`, показано на следующем рисунке слева. После этого мы выполнили его эквализацию. Новое эквализованное изображение показано справа.

Его гистограмма показана на следующем рисунке слева. Функция преобразования, построенная по гистограмме последнего изображения, имеет очень крутой подъем, однако по сравнению с функцией преобразования, приведенной нами ранее, она сдвинута в более темную часть.



2.2 Пространственная фильтрация.

Как уже было отмечено ранее, пространственная фильтрация состоит из выполнения для каждой точки (x y) изображения некоторой операции, которая использует лишь значения яркости пикселов в фиксированной окрестности вокруг этой точки. Для такой процедуры принято использовать термины окрестностная обработка или пространственная фильтрация

Если операции, совершаемые над пикселями окрестности, являются линейными, то вся процедура называется линейной пространственной фильтрацией, в противном случае она называется нелинейной пространственной фильтрацией.

Линейные операции состоят из умножения каждого пиксела окрестности на соответствующий коэффициент и суммирование этих произведений для получения результирующего отклика в каждой точке (*x, y*). Если окрестность имеет размер `m x n`, то потребуется $m \cdot n$ коэффициентов. Эти коэффициенты сгруппированы в виде матрицы, которая называется фильтром, сверточным фильтром, маской, фильтрующей маской, ядром, шаблоном или окном, причем первые три термина являются более распространенными..

Механизм линейной пространственной фильтрации заключается в перемещении базовой точки (обычно центр) фильтрующей маски `w` от точки к точке изображения `f`. В каждой точке (x, y) откликом фильтра является сумма произведений коэффициентов фильтра и величин яркостей соответствующих пикселов окрестности. Для маски размера $m \cdot n$ обычно предполагается, что m и n – нечетные положительные целые числа, чтобы в качестве базовой точки маски можно было взять ее центр.

При выполнении линейной фильтрации используется два тесно связанных понятия: корреляция и свертка. Корреляция состоит в прохождении маски `w` по изображению `f`, как об этом говорилось выше.

Свертка делается так же, но маску `w` надо повернуть на 180° перед прохождением по изображению `f`. Поясним эти понятия на простом примере.

Пусть изображение `f(x,y)` и маска `w` имеют такой вид, как показано на следующем рисунке (*а*). Начало изображения находится в верхнем левом углу матрицы `f(x,y)`. Для совершения корреляции надо поместить правую нижнюю точку `w(x,y)` так, чтобы она совпала с началом `f(x,y)`. Поскольку имеются точки, которые не перекрываются, то полагаем, что функция `f` равна нулю везде, где это необходимо, чтобы гарантировать наличие соответствующих точек на `f` при прохождении над ней маски `w` (рис. *б*). Первым значением корреляции является сумма поэлементных произведений двух функций в положении на рисунке (*в*). В этом случае сумма произведений равна 0. Затем маска `w` перемещается на один шаг вправо и процесс вычисления повторяется. Дойдя до правого края, маска сдвигается на один пиксель вниз и возвращается в крайнее левое положение. Первые 16 шагов будем получать нули, которые помещаются в точку расширенного изображения, совпадающую с центром маски.

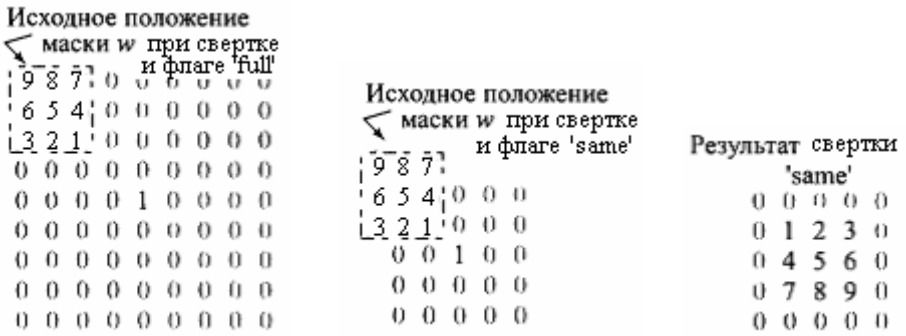
На 17 – м шаге девятка (правая нижняя точка маски `w`) совпадет с единицей (см. рис (*г*)) и мы получим первое ненулевое значение 9 x 1 = 9 корреляции. Продолжаем этот процесс до тех пор, пока `w` полностью не пройдет `f`. Мы получим результат, показанный на рисунке (*д*).

Таким образом, для совершения корреляции надо перемещать `w(x,y)` по всем возможным положениям, так чтобы хоть один пиксель маски перекрывался с пикселями изображения `f(x,y)`. Последнее вычисление производится, когда правая нижняя точка маски совмещается с правой нижней точкой расширенного изображения `f`. Отметим, что если бы мы

зафиксировали w, а перемещали бы f, то результат был бы иным, так что порядок операции здесь имеет значение.

Метка 'full' над результатом корреляции на рисунке (*д*) является флагом, который предписывает применение корреляции с расширением изображения, продемонстрированным выше методом. Можно использовать и другую опцию, 'same' (рисунок (*е*)), когда вычисляется корреляция, размер которой совпадает с размером исходного изображения f. Эти вычисления также используют расширение нулями, но при этом начальная позиция центральной точки маски (это точка с отметкой 5) совмещается с началом f. Последнее вычисление производится, когда центральная точка маски совмещается с правой нижней точкой f (нерасширенного). В результирующем изображении добавленные нули убираются, и результат совпадает по размерам с нерасширенным исходным изображением.

Для совершения свертки необходимо сначала повернуть w(x,y) на 180° вокруг центра и совершить ту же процедуру, что и при вычислении корреляции. Исходное расширенное изображение и положение на нем повернутой маски при флаге 'full' показано на следующем рисунке слева, в середине показано исходное положение маски при флаге 'same', результат свертки с флагом 'same' показан справа.



Свертка дает одинаковый результат независимо от того, какая из двух функций подвергается перемещению. В пакете IPT при реализации процедур свертки и корреляции всегда перемещается фильтрующая маска. Заметим также, что результаты корреляции и свертки получаются друг из друга поворотом на 180°.

Функция f на предыдущих рисунках являлась дискретным единичным импульсом, т. е. она равна 1 только в одной точке, а во всех остальных она равна 0. Из приведенных рисунков ясно видно, что свертка просто «копирует» маску w в то место, где был единичный импульс. Это простое свойство копирования является фундаментальной концепцией теории линейных систем и объясняет необходимость поворота одной из функций на 180° при выполнении операции свертки. Если маска является симметричной, то корреляция и свертка дают одинаковые результаты.

В пакете IPT линейная пространственная фильтрация реализована функцией imfilter, которая имеет следующий синтаксис:

```
g=imfilter(f,w,filtering_mode,boundary_options,size_options);
```

где f — это входное изображение, w — фильтрующая маска, g — результат фильтрации. Параметр filtering_mode (мода фильтрации) определяет, что совершает фильтр, корреляцию ('corr') или свертку ('conv'). Опция boundary_options (способ расширения границ) отвечает за расширение границ, причем размеры расширения определяются размерами фильтра. Этот параметр принимает числовое значение, которым следует расширить изображение. По умолчанию этот параметр равен нулю (без апострофов). Он может также принимать следующие текстовые значения:

'replicate'	Размер изображения увеличивается повторением величин на его боковых границах
'symmetric'	Размер изображения увеличивается путем зеркального отражения через границы
'circular'	Размер изображения увеличивается периодическим повторением двумерной функции

Опция size_options (опция размера) — это либо 'full', либо 'same', смысл которых уже объяснялся ранее. При опции 'full', выход имеет размеры расширенного изображения, а при опции 'same' выход имеет те же размеры, что и вход (опция по умолчанию).

Чаще всего функция imfilter применяется в виде команды

```
g = imfilter(f, w, 'replicate');
```

Эта форма команды используется при реализации в IPT стандартных линейных пространственных фильтров.

Методику пространственной фильтрации проиллюстрируем примерами.

Пример. На следующем рисунке слева показано изображение класса uint8, размера 512x512 пикселов. Преобразуем его в изображение класса double

```
f=imread('Board.tif'); imshow(f); set(gcf,'Color',[1 1 1]);
e=im2double(f); imshow(e); % то же изображение, но класса double
w = ones(31); gd = imfilter(e,w); imshow(gd, [ ]);
```

Справа показан результат применения к нему простого фильтра размера 31 x 31, который пропорционален фильтру усреднения. Фильтр усреднения получится, если единицы разделить на 31^2. Заметим, что здесь существенно то, что мы преобразовали изображение к классу double, иначе результат был бы другим. Свертка с фильтром w дает эффект размытия исходного изображения. Поскольку фильтр симметричен, мы могли использовать моду

корреляции, заданную в `imfilter` по умолчанию. Здесь также использовалась граничная опция, принятая по умолчанию, т. е. расширение изображения нулями (черным цветом). Как и ожидалось, границы между белыми и черными областями стали размытыми. Тот же эффект имел место на границах изображения, к которым примыкают белые области, что вполне понятно в силу расширения границ черным цветом.

С размытием границ можно справиться, если использовать опцию '`replicate`' (см. след. рисунок слева). Видно, что теперь границы фильтрованного изображения выглядят так, как и ожидалось. Эквивалентный результат получается с опцией '`symmetric`'. Однако, если применить опцию '`cirular`', то получим такой же результат как на предыдущем рисунке справа. Обнаруживается та же проблема, что и при расширении нулями. Этому не стоит удивляться, ведь при периодическом повторении исходного изображения черные квадраты примыкают к белым.

```
gr = imfilter(e, w, 'replicate'); imshow(gr, [ ]);
gs = imfilter(e, w, 'symmetric'); imshow(gs, [ ]);
gc = imfilter(e, w, 'circular'); imshow(gc, [ ]);
```

Проиллюстрируем эффекты, происходящие из-за сохранения класса изображения `uint8`, возникающие при применении функции `imfilter`. Если на это не обращать внимания, то это может привести к определенным проблемам. Выполним команду

```
gr = imfilter(f, w, 'replicate'); imshow(gr, [ ]);
```

Она приведет к построению изображения показанного на предыдущем рисунке справа с белыми областями в середине между квадратами. Здесь при конвертации выходного изображения в класс `uint8` функцией `imfilter` (функция `imfilter` оставляет класс исходного изображения) произошло усечение части изображения. Причина этого состоит в том, что сумма коэффициентов фильтрации не принадлежит интервалу [0,1], в результате чего фильтрованные коэффициенты вышли за допустимые пределы области определения [0, 255] данного класса. Чтобы избежать такой неприятности, следует делать нормировку коэффициентов фильтра, чтобы их сумма принадлежала интервалу [0,1] (в рассмотренном примере для этого необходимо разделить коэффициенты на 31^2, тогда их сумма будет равна 1), или переводить изображение в класс `double` как мы это сделали выше. Однако, даже в этом случае данные придется где-то перенормировать для приведения их к нужному формату, например, при сохранении изображения

на диск. В любом случае, следует всегда помнить об области значений данных, чтобы избегать неожиданных результатов.

Фильтр `w`, который мы использовали в предыдущем примере, создан нами в виде квадратной матрицы. Для выполнения операций фильтрации в MatLab имеется несколько стандартных пространственных фильтров. Их можно получить из функции `fspecial`, которая генерирует маску фильтра `w` при выполнении команды

```
w = fspecial('type', parameters),
```

где '`type`' обозначает тип фильтра, а в аргументах `parameters` задаются параметры выбранного фильтра. Этот параметр может принимать значения: '`average`' – прямоугольный усредняющий фильтр, '`disk`' – круговой усредняющий фильтр, '`laplacian`' – фильтр Лапласа `3 x 3` и некоторые другие, с которым можно познакомиться по справочной системе.

Одним из стандартных является фильтр Лапласа. Проиллюстрируем использование функций `fspecial` и `imfilter` при улучшении изображения фильтром Лапласа. Оператор Лапласа непрерывного изображения `f(x,y)` обозначается $\Delta f(x,y) = \nabla^2 f(x,y)$ и задается формулой

$$\nabla^2 f(x,y) = \frac{\partial^2 f(x,y)}{\partial x^2} + \frac{\partial^2 f(x,y)}{\partial y^2}.$$

В качестве дискретных приближений вторых производных используются выражения

$$\frac{\partial^2 f}{\partial x^2} = f(x+1,y) + f(x-1,y) - 2f(x,y)$$

$$\frac{\partial^2 f}{\partial y^2} = f(x,y+1) + f(x,y-1) - 2f(x,y),$$

Поэтому

$$\nabla^2 f = [f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1)] - 4f(x,y).$$

Это выражение можно применить в любой точке (*x, y*) изображения, сделав свертку со следующей пространственной маской (слева). Другое приближение вторых производных учитывает значения диагональных элементов и дает маску показанную справа.

$$\begin{matrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{matrix} \qquad \begin{matrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{matrix}.$$

Производные иногда определяются с противоположным знаком, что приводит к смене знаков в приведенных выше формулах масок. Улучшение изображений с помощью оператора Лапласа производится по формуле

$$g(x,y) = f(x,y) + c\nabla^2 f(x,y),$$

где `f(x,y)` — это исходное изображение, `g(x,y)` — улучшенное изображение, а параметр *c* равен 1, если центральный коэффициент маски положителен, и $c = -1$ в противном случае. Поскольку оператор Лапласа

является дифференциальным, он повышает резкость изображения, но переводит области с постоянными значениями яркости в 0. Добавление исходного изображения восстанавливает тональность уровней таких областей.

Функция `fspecial('laplacian', alpha)` реализует более сложную маску Лапласа вида

$$\begin{matrix} \frac{\alpha}{1+\alpha} & \frac{1-\alpha}{1+\alpha} & \frac{\alpha}{1+\alpha} \\ \frac{1-\alpha}{1+\alpha} & \frac{-4}{1+\alpha} & \frac{1-\alpha}{1+\alpha} \\ \frac{\alpha}{1+\alpha} & \frac{1-\alpha}{1+\alpha} & \frac{\alpha}{1+\alpha} \end{matrix}$$

которая позволяет тоньше подстраивать результат. Однако преобладающее использование оператора Лапласа основано на приведенных выше двух масках.

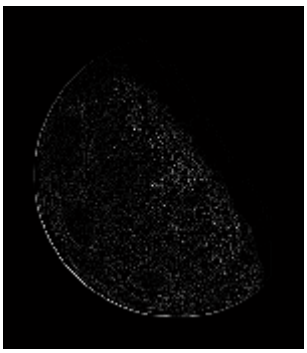
Пример. Рассмотрим улучшение изображения с помощью фильтра Лапласа. На следующем рисунке слева дан несколько расплывчатый снимок Луны. Улучшение в этом случае заключается в подчеркивании перепадов уровней яркости на изображении при сохранении, насколько это возможно, областей постоянной тональности. Сначала мы генерируем и отображаем фильтр Лапласа:

```
w = fspecial('laplacian',0)
w =
     0     1     0
     1    -4     1
     0     1     0
```

Заметим, что фильтр принадлежит классу `duoble`, и параметр формы `alpha = 0` приводит к описанному выше фильтру. Такую форму можно легко задать вручную командой `w = [0 1 0; 1 -4 1; 0 1 0];`

Теперь применим фильтр `w` к изображению `f`, которое имеет класс `uint8`:

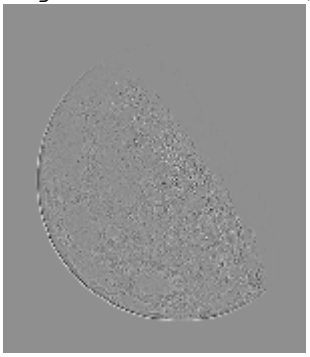
```
f=imread('Moon2.tif'); imshow(f);            % рисунок слева
g1 = imfilter(f,w,'replicate');imshow(g1,[]); % рис. справа
```



Здесь приведен результат этих команд. Он представляется правдоподобным, но имеется одна проблема. В силу отрицательности центрального коэффициента фильтра можно ожидать появления фильтрованного изображения с отрицательными значениями пикселов. Однако в нашем случае исходное изображение было класса `uint8`, а, как уже говорилось

ранее, фильтрация функцией `imfilter` создает на выходе изображение того же класса, что было на входе, поэтому отрицательные величины будут обрезаны. Чтобы обойти эту проблему, следует преобразовать изображение `f` в класс `double` перед фильтрацией:

```
f2=im2double(f);g2=imfilter(f2,w,'replicate');imshow(g2,[ ]);
```




Результат, показанный на предыдущем рисунке слева, является правильно обработанным изображением при фильтрации с помощью лапласиана.

Наконец, для восстановления тонов областей, потерянных при выполнения фильтрации лапласианом, следует вычесть (напомним, что центральный коэффициент фильтра отрицателен) отфильтрованное изображение из исходного:

```
g = f2 - g2; imshow(g);
```

Окончательный результат показан на предыдущем рисунке справа. Видно, насколько улучшенное изображение является более резким по сравнению с исходным

Задача улучшения изображений часто требует подбора параметров фильтров из имеющегося набора. Лапласиан является хорошим примером. В пакете имеется фильтр-лапласиан 3 x 3 с числом – 4 в центре. Как правило, еще большую резкость изображения можно получить с лапласианом, в центре которого стоит число – 8, окруженное со всех сторон 1. Целью следующего примера является ручная реализация этого фильтра для сравнения.

Пример. Выполним следующую цепочку команд:

```
f=imread('Moon2.tif'); imshow(f);   % исходный снимок слева
w4 = fspecial( 'laplacian', 0);
fd=im2double(f); gf4=imfilter(fd,w4,'replicate');
g4 = fd – gf4; figure, imshow(g4);   % снимок в центре
w8=[1 1 1; 1 -8 1; 1 1 1];
g8 = fd - imfilter(fd, w8, 'replicate');
figure, imshow(g8);                  % снимок справа
```



На рисунке справа показано результирующее очень резкое изображение.

Нелинейная пространственная фильтрация также основана на окрестностных операциях, причем механизм определения окрестности размера `m x n` и скольжения ее центра по изображению является таким же, как и в линейной фильтрации. Однако там, где линейная фильтрация использует сумму произведений (т.е. линейную операцию), нелинейная фильтрация основана на нелинейных операциях, совершаемых над пикселями текущей окрестности. Например, если положить, что отклик фильтра в каждой центральной точке равен максимальному значению в ее окрестности, то это определит нелинейную операцию фильтрации. Другое фундаментальное отличие состоит в том, что концепция маски не превалирует в нелинейных процессах. Идея фильтрации остается, но сам «фильтр» следует представлять себе в виде нелинейной функции, которая применяется к пикселям окрестности.

Наиболее употребительные нелинейные фильтры порождаются в пакете IPT функцией `ordfilt2`, которая строит фильтры порядковых статистик (их также называют ранговыми фильтрами). Отклики этих нелинейных пространственных фильтров основаны на предварительном упорядочении (ранжировании) пикселов изображения из текущей окрестности, после чего центральному пикселу присваивается значение, определенное в результате данного упорядочения. Синтаксис функции `ordfilt2` имеет следующий вид:

```
g = ordfilt2(f, order, domain);
```

Эта функция создает выходное изображение `g`, заменяя каждый элемент `f` на элемент с номером `order` в упорядоченной последовательности ненулевых элементов из окрестности, заданной параметром `domain`. Здесь `domain` — это матрица `m x n`, состоящая из нулей и единиц, которые обозначают позиции пикселов, участвующие в вычислениях. В этом смысле матрица `domain` действует как маска. Пикселы окрестности, которым соответствуют нули в `domain`, не участвуют в вычислениях. Например, чтобы реализовать фильтр минимума размера `m x n`, применяется команда

```
g = ordfilt2(f, 1, ones(m, n));
```

В такой записи `order=1` означает первый элемент в упорядоченной последовательности из `m·n` пикселов, а `ones(m,n)` — это матрица `m x n`, из одних единиц, указывающая на то, что все элементы окрестности участвуют в вычислении отклика фильтра. Аналогично, фильтр максимума реализуется командой

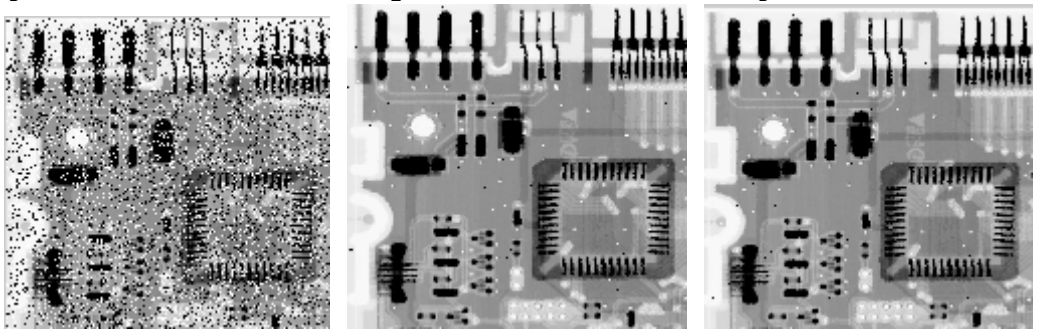
```
g = ordfilt2(f, m*n, ones(m, n));
```

Самым известным фильтром порядковых статистик в цифровой обработке изображений является медианный фильтр, который соответствует среднему номеру элементов маски. Для этого используют функцию `median` в `ordfilt2` следующим образом

```
g = ordfiIt2(f, median(1:m*n), ones(m, n));
```
Здесь `median(1:m*n)` — это медиана упорядоченной последовательности чисел 1,2,..., `m·n`. Функция `median` имеет следующий общий синтаксис:
```
v = median(A, dim),
```
где `v` — это вектор, элементы которого образуют медиану `A` вдоль размерности `dim`. Например, если `dim = 1`, то каждый элемент `v` — это вектор – строка средних значений элементов вдоль соответствующих столбцов матрицы `A`.

В силу практической важности медианного фильтра в пакете IPT предусмотрена специальная реализация этого фильтра:
```
g = medfilt2(f, [m, n], padopt),
```
где пара `[m,n]` задает окрестность размера `m x n`, по которой вычисляется медиана. Параметр `padopt` задает три возможные опции расширения границ изображения. Она может принимать значение по умолчанию '`zeros`' с нулевым расширением. Второе значение '`symmetric`', расширяет изображение `f` путем его зеркального отражения через границы. Третье значение '`indexed`', при котором f расширяется значением 1, если `f` имеет класс `double` и значением 0 в противном случае. В простейшем виде эта функция вызывается командой
```
g = medfilt2(f);
```
При этом используется окрестность 3x3 для вычисления медианы, и входное изображение расширяется нулевыми значениями пикселов.

Медианная фильтрация весьма эффективна при удалении с изображений импульсных шумов. Борьбу с различными шумами мы будем обсуждать позже, однако здесь уместно проиллюстрировать эти действия с помощью медианного фильтра.

Пример. На следующем рисунке слева представлен рентгеновский снимок f печатной платы, полученный на стадии автоматического контроля продукции производства, который сильно подпорчен шумом. Выполним следующие команды
```
f=imread('ChkBoardNoise.tif'); imshow(f);     % фото слева
gm = medfilt2(f) ; imshow(gm);                % фото в центре
gms = medfilt2(f, 'symmetric'); imshow(gms); % фото справа
```


Правое изображение похоже на среднее, но эффект почернения границ (черные точки на границах) на нем не проявляется.

Другим важным применением фильтров является расфокусировка изображения, позволяющая создать грубый образ объектов, которые могут представлять интерес. При этом мелкие объекты смешиваются с фоном, в то время как большие объекты остаются в виде пятен и могут быть легко обнаружены. Размеры объектов, которые будут смешиваться с фоном, приблизительно совпадают с размерами маски сглаживающего фильтра.

Литература.

1. Р. Гонсалес, Р. Вудс Цифровая обработка изображений. М.: Техносфера, 2005.
2. Р. Гонсалес, Р. Вудс С. Эддинс Цифровая обработка изображений в среде Matlab. М.: Техносфера, 2006.
3. В.Т. Фисенко, Т.Ю. Фисенко, Компьютерная обработка и распознавание изображений: учеб. пособие. - СПб: СПбГУ ИТМО, 2008. –192 с.
4. Яне Б. Цифровая обработка изображений. Москва: Техносфера, 2007. - 584с.