Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ
ГИДРОМЕТЕОРОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ

Е.А.Чернецова

Визуализация инженерных расчетов в Матлаб

Санкт-Петербург
РГГМУ
2024

УДК 004.6(075.8)
ББК 32.97я73

Ч49

Рецензент:

– В.Г. Бурлов, доктор технических наук, профессор Санкт-Петербургского политехнического университета (СПбПУ).

Чернецова Е.А.

Ч-49 Визуализация инженерных расчетов в Матлаб. Учебное пособие / Чернецова Е.А. – Санкт-Петербург : РГГМУ, 2024. – 90 с.

Учебное пособие «Визуализация инженерных расчетов в Матлаб» предназначено для студентов, обучающихся по специальности «Информационная безопасность телекоммуникационных систем», а также для студентов специальности «Морские информационные системы и оборудование». В учебное пособие включены теоретические сведения, практические задачи и упражнения, в которых рассматриваются средства, предоставляемые пакетом прикладных программ Матлаб в области графической обработки данных, а именно возможности визуализации результатов инженерных расчетов с помощью построения двумерных и трехмерных графиков, обработка изображений, создание стандартного графического интерфейса пользователя с помощью специальных средств Матлаб и нестандартного графического интерфейса пользователя с помощью команд Матлаб, а также использование графики для защиты информации (стеганография).

Учебное пособие предназначено для студентов гидрометеорологического университета и может быть полезным для всех желающих ознакомиться с основами теории и практики графической обработки данных в Матлаб.

УДК 004.6(075.8)
ББК 32.97я73

Введение

Модульный подход к моделированию достаточно глубоко реализуется в современных интегрированных средах (пакетах прикладных программ).

Одним из них является интегрированная среда (в определенном смысле энциклопедия технических применений при проектировании и моделировании) MATLAB (MATrix LABoratory) фирмы "The MathWorks Inc" (USA)[1].

Система MATLAB предназначена для выполнения инженерных и научных расчетов и высококачественной визуализации получаемых результатов. Эта система применяется в математических рачетах, вычислительных экспериментах, имитационном моделировании, финансовых расчетах и ряде других областей.

Входной язык программирования MATLAB близок по синтаксису к современным системам программирования на базе универсальных алгоритмических языков, таких как FORTRAN, C или Java. И хотя MATLAB уступает этим языкам в быстродействии, содержащиеся в нем уникальные библиотеки численных методов и средств визуализации результатов вычислений зачастую позволяют пользователю получить конечный результат быстрее, чем при использовании традиционных языков программирования. В пакет входит множество хорошо проверенных численных методов (решателей), операторы графического представления результатов, средства создания диалогов.

Система MATLAB обладает возможностями как процедурного, так и объектно-ориентированного языка программирования. Кроме того, она располагает средствами интерактивной разработки графических интерфейсов пользователя (GUI). Пользователь может выбрать полуавтоматический способ создания GUI с помощью инструмента GUI Layout Editor (команда guide в консоли Matlab) или «ручной» программный способ создания GUI (programmatically). Таким образом, помимо богатых возможностей языка MATLAB для создания алгоритмов, в нем содержатся также широкие возможности для проектирования пользовательского интерфейса.

Для обработки изображений пакет прикладных программ Matlab обладает тулбоксом (набором инструментов) Image Processing Toolbox.

MATLAB дает возможность инженерам и ученым легко реализовывать свои идеи. Мощные численные методы и графические возможности позволяют проверять предположения и новые идеи, а интегрированная среда дает возможность быстро получать практические результаты.

Часть I. Визуализация инженерных расчетов

1.1.Элементарные графические функции системы Matlab

Рассмотрим основные модели задания цвета в Matlab. В нем используются общепринятые в компьютерной графике модели[2]. Цвет в компьютерной графике нужен для того, чтобы:

– передать определенную информацию об объектах;

– отразить различия выводимых объектов;

– с помощью контраста вывести одни части изображения на первый план, другие же увести в фон, то есть акцентировать внимание на важном, композиционном центре;

– без увеличения размера при помощи цвета передать некоторые детали изображения;

– при помощи цвета, точнее оттенков серого, сымитировать (передать) в двумерной графике объем объекта;

– привлечь внимание зрителя, созданием красочного и интересного изображения.

Максимальное количество цветов, которое может быть использовано в изображении данного типа, называется глубиной цвета. Кроме полноцветных, существуют типы изображений с различной глубиной цвета: черно – белые, штриховые, в оттенках серого, с индексированным цветом. Некоторые типы изображений имеют одинаковую глубину цвета, но различаются по цветовой модели.

Глубина цвета определяет количество бит информации на один пиксель. Чем больше глубина цвета, тем шире диапазон доступных цветов.

Наиболее распространены изображения с глубиной цвета:

– 1 бит – двухцветные;

– 4 бита – 16 цветов;

– 8 бит – 256 цветов;

– 16 бит – 65536 цветов;

– 24 бит – 16,7 млн. цветов.

Цветовая модель ***RGB*** – одна из наиболее распространенных и часто используемых моделей. Она применяется в приборах, излучающих свет, таких, например, как мониторы, прожекторы, фильтры и другие подобные устройства.

Данная цветовая модель базируется на трех основных цветах: Red – красном, Green – зеленом и Blue – синем. Каждая из вышеперечисленных составляющих может варьироваться в пределах от 0 до 255, образовывая разные цвета и обеспечивая, таким образом, доступ ко всем 16 млн. цветов.

Эта модель аддитивная. Слово аддитивная (сложение) подчеркивает, что цвет получается при сложении трех базовых цветов, каждый своей яркости. Цвет получается следующим образом

$$\text{Цвет} = rR + gG + bB, \qquad (1.1)$$

где r,g,b – количество основных цветов (вклад), характеризующих яркость.

Основные цвета при смешивании базовых цветов показаны на рис.1.1.1.

Рис. 1.1.1. Основные цвета при смешивании базовых цветов

Яркость каждого базового цвета может принимать значения от 0 до 255 (256 значений), таким образом, модель позволяет кодировать 256^3 или около 16,7 млн. цветов. Эти тройки базовых точек (светящиеся точки) расположены очень близко друг к другу, так что каждая тройка сливается для нас в большую точку определенного цвета. Чем ярче цветная точка (красная, зеленая, синяя), тем большее количество этого цвета добавится к результирующей (тройной) точке. При увеличении яркости отдельных составляющих будет увеличиваться и яркость результирующего цвета: если смешать все три цвета с максимальной интенсивностью, то результатом будет белый цвет; напротив, при отсутствии всех цветов получается черный.

На рис.1.1.2 показана система координат RGB – цветовой куб модели RGB. Начало отсчета (0, 0, 0), соответствует черному цвету. Максимальное значение RGB – (1, 1, 1) соответствует белому цвету.

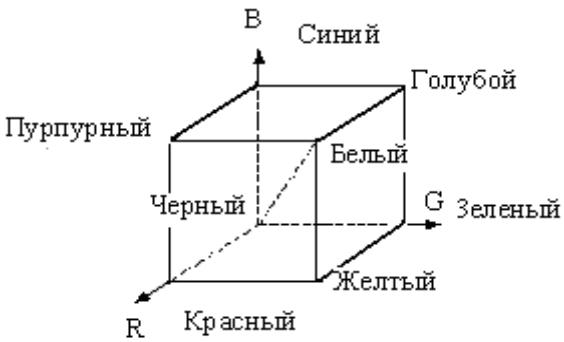


Рис.1.1.2. Цветовой куб модели R G B

Модель является аппаратно – зависимой, так как значения базовых цветов (а также точка белого) определяются качеством примененного в мониторе люминофора. В результате на разных мониторах одно и то же изображение выглядит неодинаково.

Несомненными достоинствами данного режима является то, что он позволяет работать со всеми 16 млн, а недостаток состоит в том, что при выводе изображения на печать часть из этих цветов теряется, в основном самые яркие и насыщенные, также возникает проблема с синими цветами.

Изображение в оттенках серого – это изображение, в котором значение каждого пикселя – это единственный образец , представляющий только количество света ; то есть он несет только информацию интенсивности . Изображения в градациях серого, разновидность черно–белого или серого монохромного , состоят исключительно из оттенков серого . Контраст варьируется от черного при самой слабой интенсивности до белого при самой сильной.

В Матлабе команда rgb2gray преобразует изображение в серое, с 256 оттенками серого, от 0 (чёрный) до 255 (белый). Таким образом, трехмерное изображение из цветной модели RGB становится двухмерным и может рассматриваться как матрица, в которой каждый элемент содержит информацию об интенсивности света пикселя. Поскольку пакет прикладных программ Матлаб предоставляет широкие возможности для работы с матрицами, то преобразование матриц и выполнение математических операций над ними позволяют решать задачи обработки изображений в оттенках серого цвета.

В связи с вышесказанным, рассмотрим действия над матрицами в системе Матлаб и возможности визуализации полученных результатов[3].

В рабочем окне MATLAB можно задавать переменные, представляющие собой числа, векторы–строки (фактически представляющие собой матрицу размером $[1\times m]$), векторы-столбцы (фактически представляющие собой матрицу размером $[1\times n]$) и матрицы размерностью $[m\times n]$).

Например, вводим переменную A=2. В рабочем окне выводится:

» A=2

A =

2

Если на конце выражения поставить знак «;» например:

A=2;

то выражение будет сразу записано в память без вывода на экран. Это позволяет сокращать время выполнения вычислительных программ и операций.

Числа в обозначении вектора–столбца отделяются друг от друга точкой с запятой, числа в обозначении вектора–строки отделяются друг от друга запятой, запятая в обозначении десятичных дробей обозначается точкой. При вводе векторов и матриц используются квадратные скобки.

Пример ввода в рабочем окне MATLA вектора–столбца размерностью $[5\times1]$:
A=[1;12;34;4.768;5.13]

A =

1.0000
12.0000
34.0000
4.7680
5.1300

Пример ввода в рабочем окне MATLAB вектора–строки размерностью $[1\times3]$:

» *A*=[1.25,3.567,8.74]

A =

1.2500 3.5670 8.7400

Пример ввода в рабочем окне MATLAB матрицы размерностью [3 x 3]:

» *A*=[1,2.3,5.2;2,4,8;9.3,3.7,2.56]

A =

1.0000 2.3000 5.2000
2.0000 4.0000 8.0000
9.3000 3.7000 2.5600

Формат представления чисел на экране дисплея по желанию пользователя можно регулировать с помощью команды format. В системе MATLAB возможно задание нескольких выходных форматов. В данных примерах используется formatshort вывода чисел на экран (5 значащих десятичных цифр), можно установить formatlong (16 значащих десятичных цифр).

Формирование векторов и подматриц. Оператор «:» является очень полезным оператором языка MATLAB. Он применяется для формирования векторов и матриц или для выделения из них подвекторов, подматриц, подблоков массива.

Например, зададим вектор *A*, состоящий из цифр от 1 до 10 следующим образом:

A=1:10

A =

1 2 3 4 5 6 7 8 9 10

Зададим вектор *A*, состоящий из цифр от 1 до 10 с шагом 2 следующим образом:

A=1:2:10

A =

1 3 5 7 9

Зададим матрицу А следующим образом:

A =

1 2 3 4 5

2 3 4 6 7

4 6 8 9 0

1 3 6 9 9

3 5 6 7 8

1 2 3 4 6

и выделим из нее подблок, включающий в себя строки со второй по четвертую, столбцов с третьего по пятый следующим образом:

A(2:4,3:5)

ans =

4 6 7

8 9 0

6 9 9

Оператор *A*(5, :) позволяет выделить все столбцы пятой строки массива *A*;

A(5,:)

ans =

3 5 6 7 8

Оператор A(:, 5) позволяет выделить все строки пятого столбца массива *A*;

A(:,5)

ans =

5

7

0

9

8

6

Операторы умножения «.*» и правого деления «./» с точкой используются при перемножении или делении массивов (каждое число первого массива умножается/делится на соответствующее число второго массива).

Например:

A =

1 2 3

4 5 6

1 4 8

B =

1 5 7

5 7 9

5 3 2

A.**B*

ans =

```
 1    10    21

20    35    54

 5    12    16
```

Операторы умножения и правого деления без точки применяются при перемножении или делении матриц по правилам линейной алгебры.

Например:

*A*B*

```
ans =

   26    28    31

   59    73    85

   61    57    59
```

При применении оператора левого деления с точкой «.\» и оператора без точки «\» выполняется решение систем линейных уравнений вида *AX=B* по методу наименьших квадратов для матриц и векторов. Например:

A\B

```
ans =

    6.3333  -14.3333  -23.3333

   -9.6667   25.6667   41.6667

    4.6667  -10.6667  -17.6667
```

» *A.\B*

```
ans =

    1.0000    2.5000    2.3333

    1.2500    1.4000    1.5000

    5.0000    0.7500    0.2500
```

При транспонировании массива (операция «.'») строки просто заменяются столбцами:

A.'

ans =

1 4 1

2 5 4

3 6 8

При транспонировании матрицы (операция «‘» без точки) результатом является транспонированная матрица, для комплексных чисел выполняется операция комплексного сопряжения.

Над матрицами можно производить разнообразные операции.

Например, с помощью команды diag возможно формировать или извлекать диагонали матрицы.

Функция *X*=diag(*v*) формирует квадратную матрицу *X* с вектором *v* на главной диагонали;

Функция *X*=diag(*v*,*k*) формирует квадратную матрицу *X* порядка length(*v*)+abs(*k*) с вектором *v* на *k*-той диагонали.

Функция *v*=diag(*X*,*k*) извлекает из матрицы *X* диагональ с номером *k*; при *k*>0 это номер *k*-й верхней диагонали, при *k*<0 это номер *k*-й нижней диагонали.

Функция *B*=reshape(*A*,*m*,*n*) возвращает матрицу размером [*m*x*n*], сформированную из элементов матрицы *A* путем их последовательной выборки по столбцам.

Элементарные графические функции системы MATLAB позволяют построить на экране и вывести на печать следующие типы графиков: линейный, логарифмический, полулогарифмический, полярный.

Для каждого графика можно задать заголовок, нанести обозначение осей и масштабную сетку.

Рассмотрим некоторые операторы для построения графиков в системе MATLAB.

Команда *plot(y)* строит график элементов одномерного массива *y* в зависимости от номера элемента; команда *plot*(*x*,*y)* соответствует построению обычной функции, когда одномерный массив *x* соответствует значениям аргумента, а одномерный массив *y* – значениям функции. Команда *plot*(*x*,*y*,*s*) позволяет выделить график функции, указав способ отображения линии, способ отображения точек, цвет линий и точек с помощью строковой переменной *s*, которая может включать до трех символов из следующей таблицы:

y желтый . точка - непрерывная

m фиолетовый o кружок : пунктирная

*c*голубой x крестик -. штрихпунктирная

r красный + плюс -- штриховая

g зеленый * звездочка

b синий *s* квадрат

w белый *d* ромбик

k черный: *v* треугольник (нижний)

^ треугольник (верхний)

< треугольник (левый)

> треугольник (правый)

p пентаграмма

h гексаграмма

Команда *plot*(*x1,y1,s1,x2,y2,s2,……)* позволяет объединить на одном графике несколько функций *y1*(*x1), y2*(*x2),.....,* определив для каждой из них свой способ отображения.

В системе MATLAB предусмотрено несколько команд и функций для построения трехмерных графиков. Значения элементов числового массива рассматриваются как z–координаты точек над плоскостью, определяемой координатами *x* и *y*. Возможно несколько способов соединения этих точек.

Первый из них – это соединение точек в сечении (функция *plot3*). Например, команда *plot3*(*x1,y1,z1,s1,x2,y2,z2,s2,…)* позволяет объединить на одном графике несколько функций *z1*(*x1,y1),z2(x2,y2*),…., определив для каждой из них свой способ изображения.

Команда *mesh*(*X,Y,Z)* выводит на экран сетчатую поверхность для значений массива *Z*, определенных на множестве значений массивов *X* и *Y*. Цвета узлов поверхности задаются командой *colormap*(*C),* где *C* – палитра.

Цвета:

hsv – выделение оттенками

hot – черно-красно-желто-белые цвета

gray – линейная шкала оттенков серого цвета

bone – шкала серых цветов с оттенками белого

copper – линейный медный цвет

pink – пастельные оттенки розового цвета

white – белый цвет

flag – попеременно красный, белый, синий и черный цвета

lines – линейное чередование цветов

colorcube – чередование серого, красного, зеленого и белого

vga - 16 цветов

jet – вариант hsv

prism – использование 6 цветов

cool – оттенки синего и фиолетового

autumn – оттенки красного и желтого

spring – оттенки фиолетового и желтого

winter–оттенки синего и зеленого

summer – оттенки зеленого и желтого

Команда *surf(X,Y,Z)* выводит на экран сетчатую поверхность для значений массива *Z*, определенных на множестве значений массивов *X* и *Y*.

Для вращения трехмерного изображения используется команда *rotate3d.*

Для того, чтобы построить несколько графиков и разместить каждый график в своем окне *figure*, нужно после построения первого графика выполнить команды: *file – New – Figure*, тогда будет создано новое окно, в котором будет изображен новый график после введения в рабочем окне соответствующих команд.

Для сохранения рабочей области MATLAB используется команда *save*. Команда *save имя файла* выгружает все переменные из рабочей области в двоичный файл с именем *имя файла.mat*. Если имя файла отсутствует, то ему присваивается специальное имя *matlab.mat.* Команда *save имя файла ABC* выгружает переменные *A, B, C*. Для очистки рабочей области используется команда *clearall*. Для очистки экрана используется команда *clc*. Для загрузки переменных в рабочую область используется команда *load*. Команда *load* без параметров считывает данные из файла *matlab.mat*, если он создан командой save. Команда *load имя файла* загружает переменные из mat-файла имя файла.mat. После загрузки переменных в рабочую область бывает необходимо вывести список переменных текущей рабочей области. Эта операция выполняется с помощью команды *who.*

1.2. Построение графиков функций, заданных в символьном виде

Система MATLAB сама по себе способна выполнять только численные расчеты. Поэтому если попытаемся вычислить выражение, в которое входят переменные, не имеющие числового значения, то получим сообщение об ошибке:

```
>> (x-1)^2
??? Undefined function or variable 'x'.
```

Возможность проведения символьных вычислений в MATLAB обеспечивается подключением к этой системе пакета компьютерной алгебры Maple[4]. Реализовано это подключение при помощи пакета Symbolic Math Toolbox.

Символьные операции можно выполнять только в том случае, если все операнды в выражении имеют символьный тип (sym). Приведение числовых констант и переменных к символьному типу осуществляется посредством функции sym.

```
>>syms x y
>> f = (x-y)^3
f=
(x-y)^3
>> g = expand( f )
g=
x^3-3*x^2*y+3*x*y^2-y^3
>>factor( g )
ans =
(x-y)^3
```

Функция simplify обеспечивает упрощение выражений.

```
>> h = sin(x)^4 + 2*cos(x)^2 - 2*sin(x)^2 - cos(2*x)
h=
sin(x)^4+2*cos(x)^2-2*sin(x)^2-cos(2*x)
>>simplify( h )
ans =
cos(x)^4.
```

Помимо упомянутых функций, в состав пакета Symbolic Math Toolbox входят функции, позволяющие в символьном виде решать задачи линейной алгебры, находить производные и первообразные, работать со степенными рядами, выполнять интегральные преобразования и решать системы дифференциальных уравнений.

Если функция одной переменной имеет аналитическое задание, то ее график строится с помощью процедуры *ezplot*[5]. Процедура ezplot имеет следующий синтаксис.

```
ezplot(f, [xmin,xmax,ymin,ymax])
```

Построим график функции **y = xsinx**. Функцию запишем в виде строки.

```
>>syms x
>>ezplot( x*sin(x), [-15 15] )
>>grid on
```

Результат приведен на рис.1.2.1.

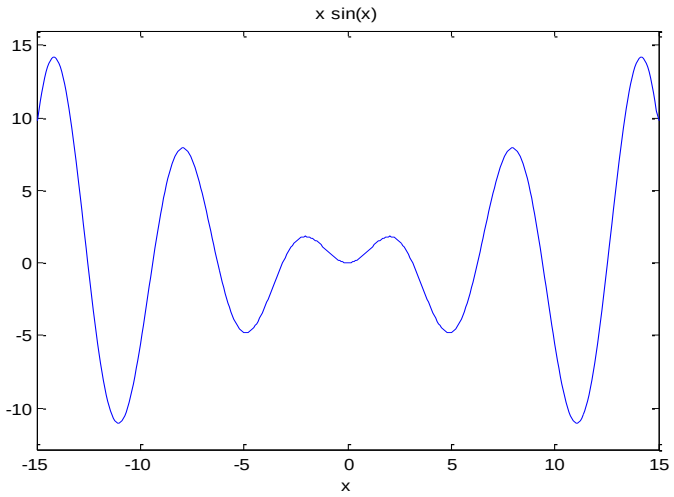


Рис. 1.2.1. График функции $y = x\sin x$

```
ezplot( exp(x^2)*sin(1/x), [-0.5 0.5 ])
>>grid on;
```

Результат приведен на рис.1.2.2.

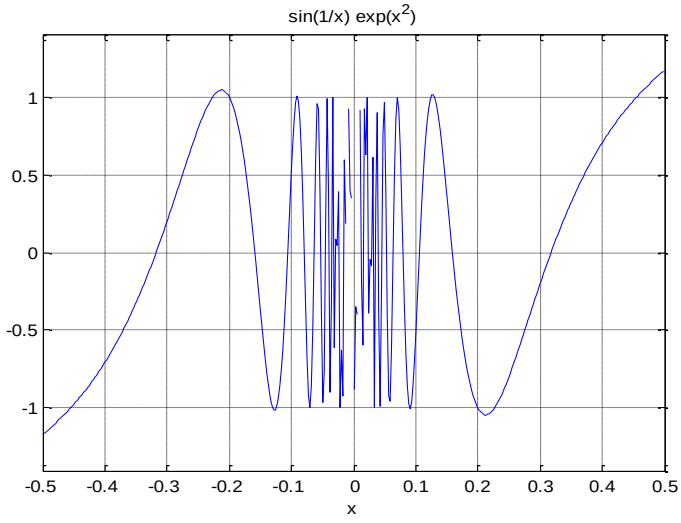


Рис.1.2.2 График функции $\sin(1/x \exp(x^2))$

Нарисуем, например, окружность, задаваемую уравнением $x^2 + y^2 = 25$.

```
syms x y
ezplot( x^2 + y^2 - 25 )
```

Чтобы окружность не выглядела как эллипс, следует выбрать одинаковый масштаб по обеим координатным осям. Это можно сделать при помощи команды axis equal.

```
>> axis equal
```

Результат приведен на рис.1.2.3.

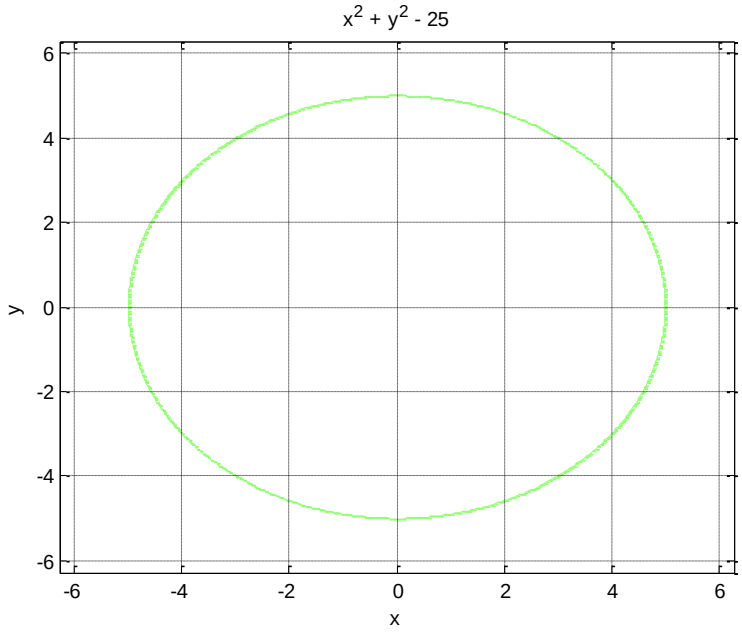


Рис.1.2.3. График функции $x^2 + y^2 = 25$

Процедура ezplot допускает также построение графиков функций, имеющих параметрическое задание, то есть таких функций, которые могут быть записаны в виде: $x = x(t)$, $y = y(t)$.

Фигуры Лиссажу в параметрической форме имеют следующую запись:

$x = \sin(mt)$, $y = \cos(nt)$.

```
syms t;
ezplot(sin(2*t), cos(3*t), [-pi pi] )
```

Результат приведен на рис.1.2.4.

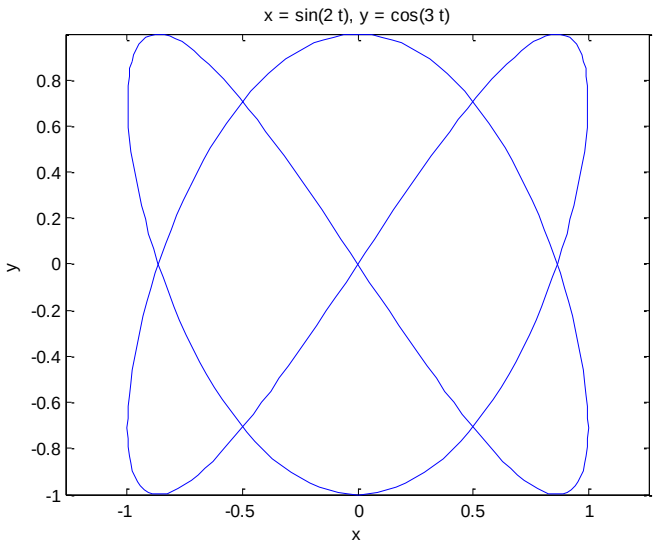


Рис.1.2.4. Фигура Лиссажу с параметрами m=2; n=3

Изобразим на графике кривую, носящую название спираль Архимеда. Эта кривая в полярных координатах записывается в виде $r = a\phi$. В нашем примере положим a =2. Symsphi. Результат приведен на рис.1.2.5.

ezpolar(2*phi, [0 4*pi])

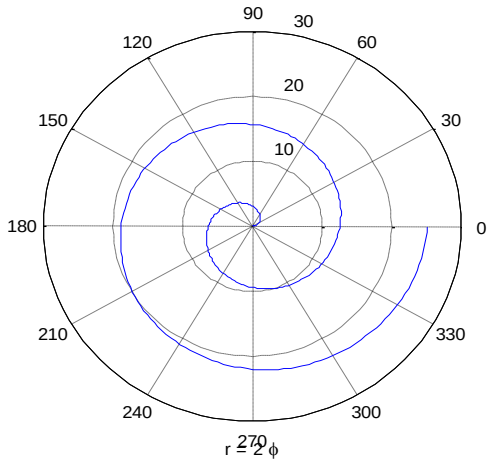


Рис. 1.2.5. Спираль Архимеда с параметром a =2

Для изображения функций, заданных параметрически, $x=f(t)$, $y=g(t)$ команда вызывается в формате ezplot('f(t)', 'g(t)', [t0, t1]).

Пусть, например, требуется нарисовать на плоскости (x, y) кривую, заданную уравнениями:

$x = 2t - 4t^3$, $y = t^2 - 3t^4$.

Набирая в командной строке код:

ezplot('2*t-4*t^3','t^2-3*t^4',[-1.5,1.5]),

получаем кривую, известную в математической теории катастроф она известна как «ласточкин хвост»- рис. 1.2.6.

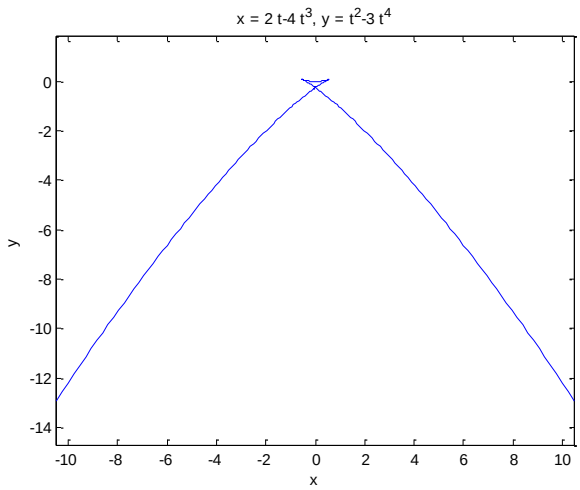


Рис. 1.2.6. Кривая «ласточкин хвост»

Если функция двух переменных имеет аналитическое задание, то ее график строится с помощью процедуры ezsurf.

ezsurf(f, [xmin, xmax, ymin, ymax])

Реализация приведена на рис. 1.2.7.

ezsurf(cos(x*y), [-3 3 -3 3])

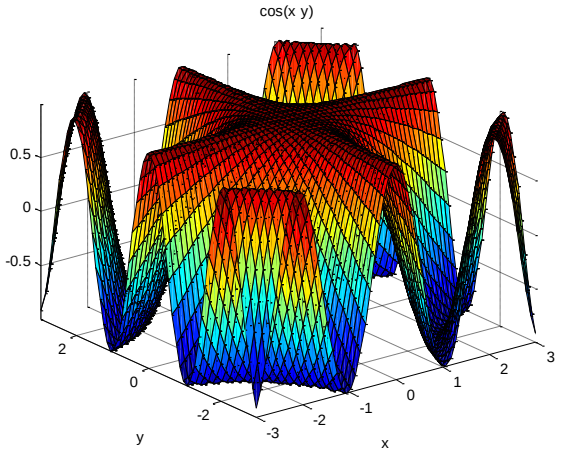


Рис. 1.2.7. Поверхность функции cos(x*y)

Каркас поверхности, образованный путем соединения ее ближайших точек отрезками прямых, строится с помощью функции ezmesh,имеющей тот же синтаксис что функция ezsurf. Реализация приведена на рис. 1.2.8.

ezmesh(cos(x*y), [-3 3 -3 3])

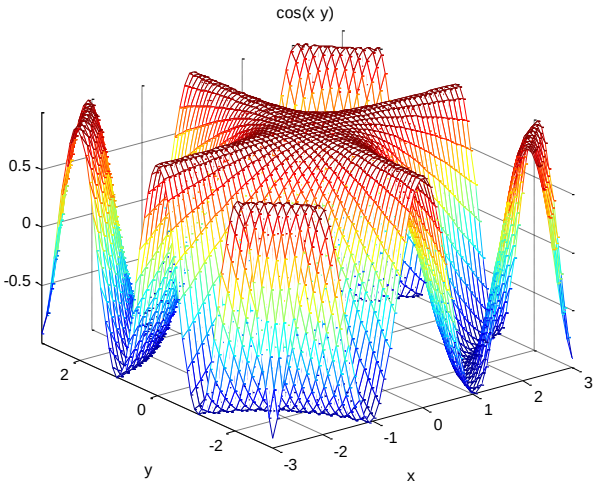


Рис. 1.2.8. Сетчатая поверхность функции cos(*x***y*)

Примеры других поверхностей приведены на рис. 1.2.9–1.2.10.

Процедуры ezsurf и ezmesh способны также строить поверхности, заданные в параметрической форме.

```
syms u v
 x = cos(u)*sin(2*v);
```

```
 y = sin(u)*sin(2*v);
 z = sin(v);
ezsurf( x, y, z, [0 2*pi 0 2*pi])
```

axisequal

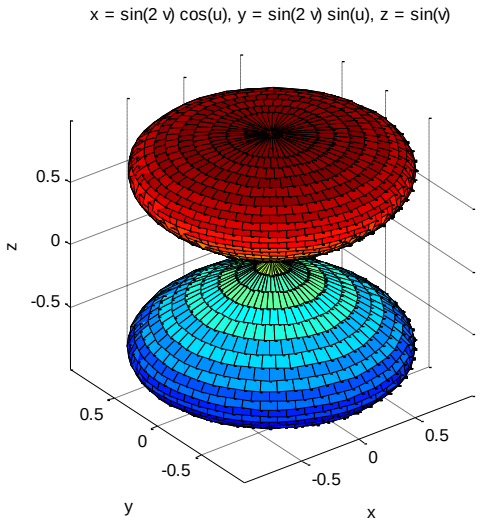


Рис. 1.2.9. Поверхность функции, заданной в параметрической форме

Графики в полярных координатах строятся с помощью процедуры polar(phi, r, format ...). Здесь phi – массив значений полярного угла, а r – массив значений радиус-вектора. Строка формата строится точно так же, как и в случае функции plot. Построим, например, красную пятилепестковую розу.

```
phi = linspace(0, 2*pi, 100);
r=2*sin(5*phi);
polar(phi, r, 'r')
```

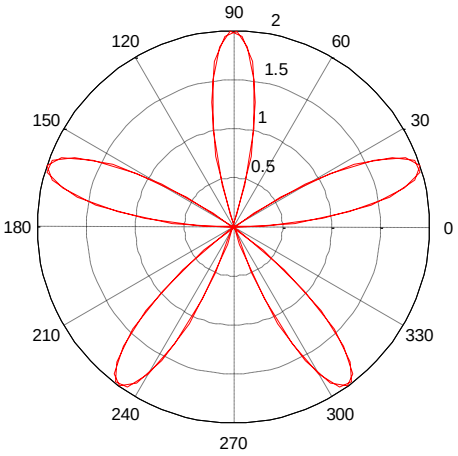


Рис. 1.2.10. Пятилепестковая «роза», построенная в полярных координатах

1.3. Построение поверхностей в цилиндрических и сферических координатах

В системе MATLAB нет процедур, предназначенных для непосредственного построения поверхностей, заданных в системах координат, отличных от декартовой. Но имеются функции, позволяющие осуществлять пересчет координат узловых точек поверхности как из сферической и цилиндрической системы координат в декартову (pol2cart и sph2cart), так и наоборот (cart2pol и cart2sph).

В качестве примера построим конус, заданный в цилиндрических координатах уравнением $r = z$.

Функция x = linspace($x1$, $x2$) формирует линейный массив размера 1 x 100, начальным и конечным элементами которого являются точки $x1$ и $x2$.

Функция x = linspace($x1$, $x2$, n) формирует линейный массив размера 1 x n, начальным и конечным элементами которого являются точки $x1$ и $x2$.

```
phi = linspace(0, 2*pi, 50);
 z = linspace(-2, 2, 50);
 [Phi, Z] = meshgrid( phi, z );
 [X_c,Y_c,Z_c] = pol2cart(Phi,Z,Z);
surf(X_c,Y_c,Z_c)
axis equal
```

Реализация приведена на рис.1.3.1.

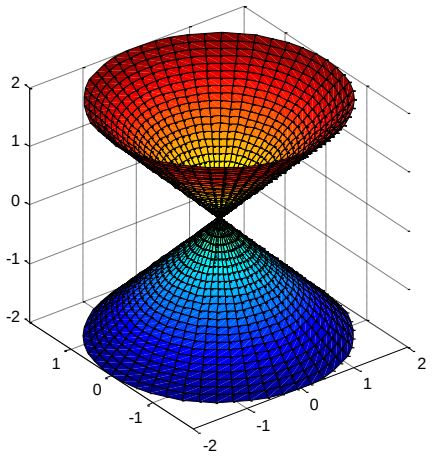


Рис. 1.3.1. Поверхность конуса, заданная в цилиндрических координатах

Построим поверхность шара, заданную в сферических координатах

```
phi = linspace(0, 2*pi, 50);
theta = linspace(0, pi, 25);
[Phi, Theta] = meshgrid( phi, theta );
```

```
[X_s,Y_s,Z_s] = sph2cart(Theta,Phi,1);
surf(X_s,Y_s,Z_s)
axis equal
```

Реализация приведена на рис.1.3.2.

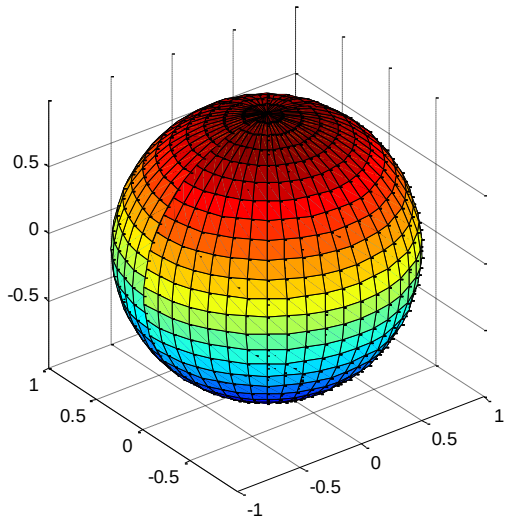


Рис. 1.3.2. Поверхность шара, заданная в сферических координатах

1.4. Вывод текста в графическое окно

Для вывода текста в графическом окне обычно используется функция gtext, позволяющая с помощью мышки поставить текст в нужную позицию. Введем команду ht = gtext('mygrafik');.Вслед за этим необходимо переместить мышь в поле графического окна и нажатием левой кнопки указать положение текста. После этого текст появится на экране в указанной позиции. Графический текст является графическим объектом со своими полями данных. Чтобы просмотреть все поля объекта и их возможные стандартные значения нужно иметь указатель на графический объект (в нашем случае указатель хранится в переменной ht) и воспользоваться командой set. Например, set(ht);.

В файле settxt.m приведен пример программы, которая позволяет с помощью мышки переместить текст в любое место на графическом окне.

Для вызова этой функции достаточно ввести в командном окне команду settxt; и указать нажатием левой кнопки мыши новое положение для текста. Функция завершает работу при нажатии правой или средней кнопок мыши. Вот текст программы **settxt.m.**

```
ht = gtext('mygrafik');
but=1;
while but==1
```

```
[xi yi but]=ginput(1);
set(ht,'position',[xi yi]);
end
```

В таблице 1.4.1 приводятся некоторые поля и их описание.

Таблица 1.4.1. Атрибуты тестового поля

Поле	Описание
color	Задает цвет текста. Например, чтобы задать зеленый цвет нужно ввести команду set(ht,'color',[0 1 0]);.
fontname	Определяет имя шрифта.
fontunits	Определяет единицы измерения размера шрифта. Возможны значения [inches | centimeters | normalized | {points} | pixels].
fontsize	Определяет размер шрифта в единицах, определенных полем fontunits.
units	Единицы для измерения поля position. Возможные значения [inches | centimeters | normalized | points | pixels | characters | {data}]. По умолчанию поле units имеет значение data, что означает что поле position определяет положение текста на экране в единицах, нанесенных на осях координат.
position	Задает положение текста в графическом окне в единицах, определенных полем units. Поле задается вектор-строкой из трех чисел [xyz]. Если график двумерный, координата z=0. Например,set(ht,'position',[1 0.5 0]);.
rotation	Определяет поворорт текста в градусах. Например, команда set(ht,'rotation',45); поворачивает текст ht на 45 градусов.
backgroundcolor	Определяет цвет фона. Например, пурпурный фон текста можно задать командой set(ht,'backgroundcolor',[1 0 1]);.
string	Строка текста. Например, set(ht,'string','newstping');.
visible	Определяет виден ли текст на экране. Возможные значения 'on' и 'off'. Например, чтобы сделать текст ht невидимым нужно ввести команду set(ht,'visible','off');.

Программа settxt получает один аргумент. При вызове функции это должен быть указатель на объект, который необходимо переместить в новую позицию. Цикл while всегда заканчивается оператором end. В нашем случае цикл while работает до тех пор пока переменная but имеет значение единица. Функция ginput с аргументом 1 останавливает работу цикла и ожидает ввода мыши в поле графического окна и однократного нажатия

(аргумент 1) любой из кнопок мыши. После этого она возвращает координаты xi и yi, где была нажата кнопка и номер нажатой кнопки в переменную but. Левая кнопка соответствует but=1, средняя but=2 и правая but=3.

1.5. Создание контурных графиков

В ряде случаев, например, при поиске локальных максимумов и минимумов, значительную информацию о функции двух переменных можно получить, построив ее контурный график. Линии уровня, образующие такой график, являются проекциями на плоскость XOY кривых, полученных в результате пересечения поверхности z = f(x,y) с плоскостями z =const.

Для построения контурных графиков функций предназначены процедуры ezcontour и contour. Первая из них используется в случае аналитического, а вторая – в случае табличного задания исследуемой функциональной зависимости. Синтаксис этих процедур аналогичен синтаксису процедур ezsurf и surf.

ezcontour(f, [xmin, xmax, ymin, ymax])

Наберем в рабочей области команду: ezcontour('-5*x/(x^2 + y^2 + 1)', [-3 3 -3 3]). Результат представлен на рис.1.5.1

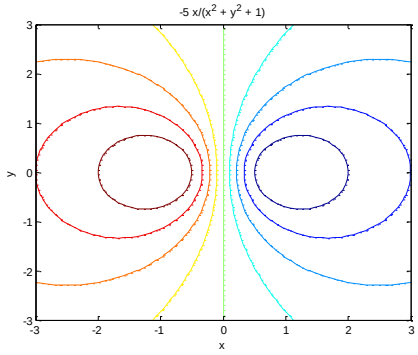


Рис. 1.5.1 Контурный график функции -5*x/(x^2 + y^2 + 1)

Указав в процедуре contour четвертый аргумент, можно задать число линий уровня, которые необходимо построить. Цвет каждой линии контурного графика выбирается из таблицы цветов в соответствии со значением функции на этом уровне. Таблицу цветов, так же как и в случае пространственного графика, можно сменить командой colormap. Линейка, показывающая соответствие между цветами и значениями функции, выводится с помощью команды colorbar.

```
[X, Y] = meshgrid( [-3 :0.1: 3] );
 Z = -5*X./(X.^2 + Y.^2 + 1);
```

contour(X, Y, Z, 30);

colormap jet

colorbar

Результат представлен на рис.1.5.2

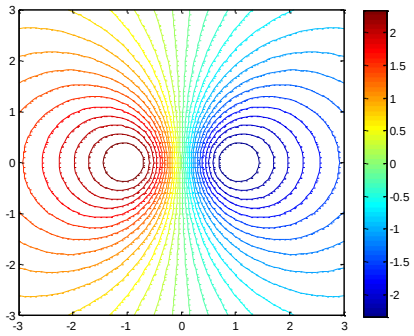


Рис. 1.5.2 Контурный график функции с линейкой цветов

Закрасить в разные цвета области, расположенные между соседними линиями уровня, можно с помощью процедуры contourf.

contourf(X, Y, Z, 15); colormap hot

Результат представлен на рис.1.5.3

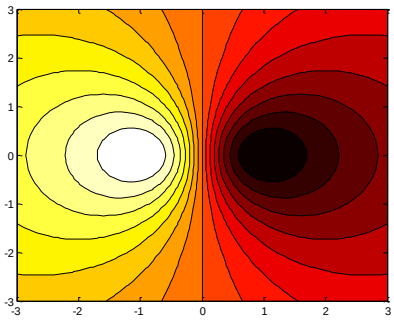


Рис.1 5.3. Контурный график функции с закрашиванием областей

По умолчанию линии уровня на контурном графике не снабжаются какими-либо обозначениями. Выполнить оцифровку такого графика можно, воспользовавшись функцией clabel, размещающей на каждой линии числовое значение уровня. Функции clabel необходимо передать в качестве аргументов два массива, содержащих информацию о линиях уровня. Требуемые массивы могут быть получены с помощью процедуры contour. Например. запись

[C, h]=contour(X, Y, Z);

clabel(C, h)

дает результат, представленный на рис.1.5.4.

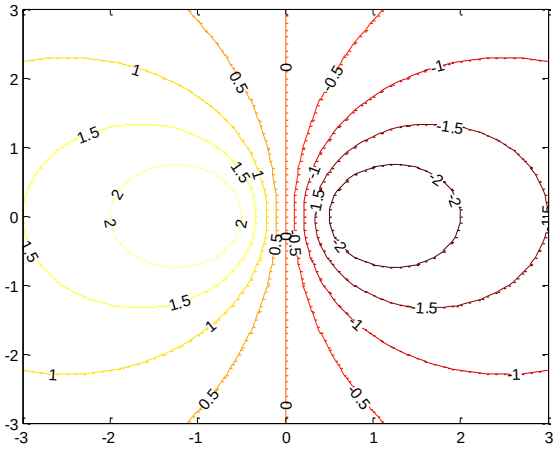


Рис.1 5.4. Контурный график функции с оцифровкой графика

В MATLAB имеются процедуры meshc, surfc, ezmeshc, ezsurfc, позволяющие объединить на одном рисунке и график поверхности заданной функции, и ее линии уровня. Например, запись

ezsurfc('(x^2-y)/(1 + x^2 + y^2)', [-pi pi -pi pi])

дает результат, представленный на рис.1.5.5.

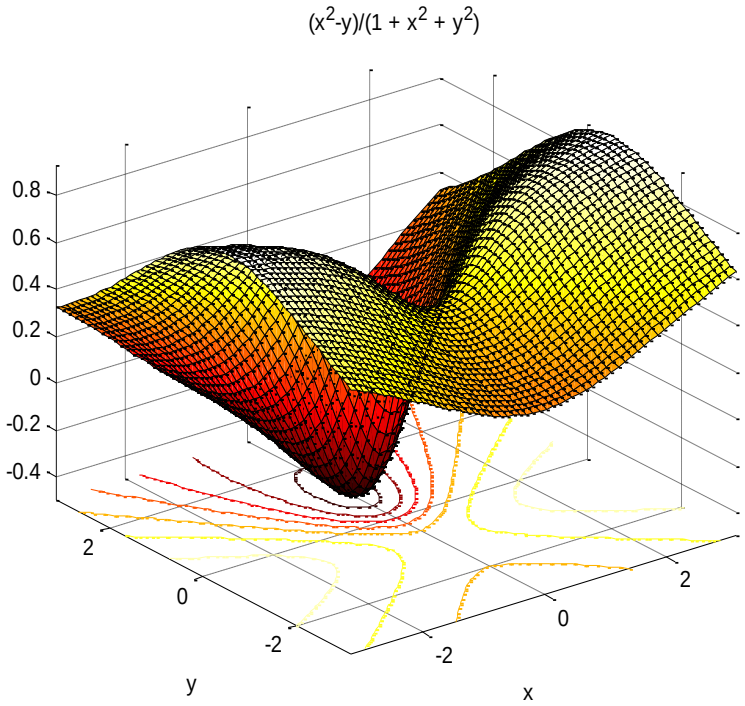


Рис.1 5.5. График поверхности и ее контур, расположенные на одной фигуре

1.6. Рисование пространственных кривых

Чтобы построить в MATLAB пространственную кривую, следует прибегнуть к помощи функций plot3 и ezplot3.

Процедура ezplot3 позволяет нарисовать пространственную кривую, имеющую параметрическое задание: ($x = x(t)$, $y = y(t)$, $z = z(t)$, $t \in$ [tmin, tmax].

Синтаксис команды: ezplot3(x, y, z, [tmin tmax]). Например, запись

```
ezplot3('sin(t)','cos(t)','t',[0,6*pi])
```

дает результат, представленный на рис.1.6.1.

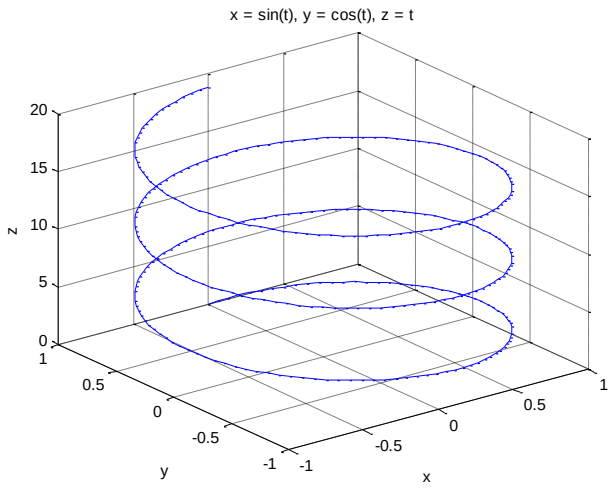


Рис.1.6.1 График пространственной кривой, имеющей параметрическое задание

Пространственная кривая, заданная массивами значений своих координат, строится с помощью функции plot3. Например, запись

```
t = linspace(0, 2*pi, 20000);
x = (2 + 4/5*cos(7*t)).*cos(4*t);
y = (2 + 4/5*cos(7*t)).*sin(4*t);
z = sin(7*t);
plot3( x, y, z, 'r', 'LineWidth', 2)
view(45,10)
```

Дает результат, представленный на рис.1.6.2.

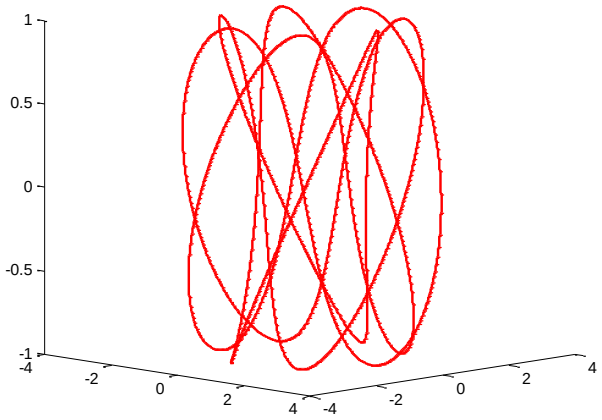


Рис.1.6.2. График пространственной кривой, заданная массивами значений своих координат

Если пространственная кривая представляет собой траекторию движения материальной точки, то для ее отображения предпочтительнее использовать процедуру comet3. В этом случае, прорисовка кривой осуществляется по мере движения точки по траектории, что позволяет лучше понять эволюцию исследуемого процесса.

comet3(x, y, z)

В MATLAB имеется аналогичная comet3 процедура comet, предназначенная для отображения движения точки на плоскости.

1.7. Вывод нескольких графиков в одно графическое окно

Иногда возникает необходимость отобразить в одном графическом окне несколько разнородных графиков. Добиться этого можно двумя способами, каждый из которых является более или менее предпочтительным в зависимости от конкретной задачи.

Первый способ заключается в использовании команды hold on, блокирующей создание нового графического окна. Таким образом, если вы построите первый график и затем выполните команду hold on, то очередной и все последующие графики будут построены в том же самом графическом окне и в тех же координатных осях. Например, запись

```
ezplot('x^2-4', [-5 5 -5 5] );
hold on
fplot('t^2*exp(-t/2)', [0 5], 'g-' );
 x = [-15:0.25:15];
 y = sin(3*x).*cos(x/3);
plot( x, y, 'r--');
```

```
axis( [-5 5 -5 5 ])
```
Дает результат, представленный на рис.1.7.1.

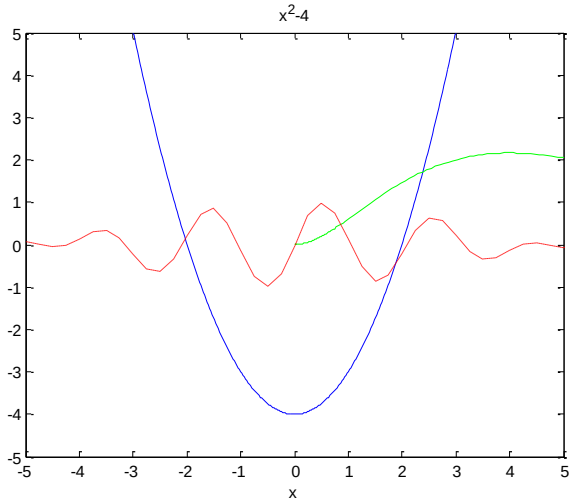


Рис.1.7.1 Вывод нескольких графиков в одно графическое окно

Отключается этот режим командой hold off.

В одни координатные оси можно помещать графики, построенные с помощью различных графических процедур, но смешивать двумерные и трехмерные графики не следует. Например, запись

```
[x, y] = meshgrid( [-5:0.5:5] );
surf(x, y, x.^2+y.^2);
hold on
ezsurf('x^2-y^2');
hold off
```
Дает результат, представленный на рис.1.7.2.

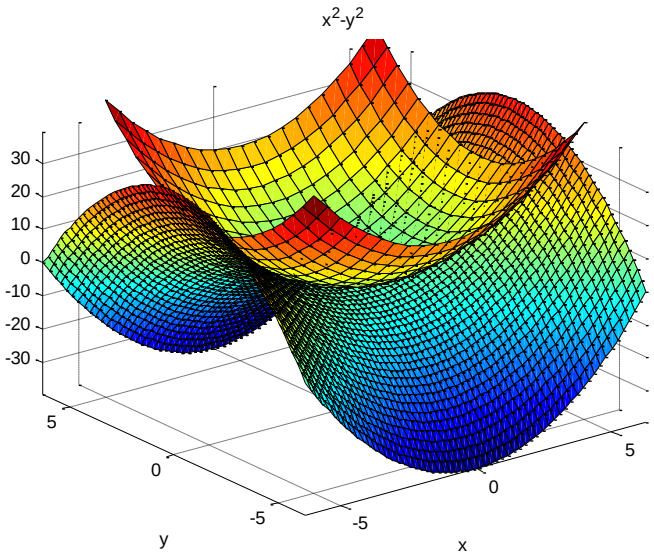


Рис.1 7.2 Вывод в одно графическое окно двух поверхностей вида x^2+y^2 и x^2-y^2

Второй способ позволяет отобразить в одном окне графики совершенно разной природы. Это становится возможным благодаря тому, что в данном случае каждый график размещается в своих собственных координатных осях.

Достигается это при помощи функции subplot, которая разбивает область графического окна на несколько прямоугольных областей равного размера, расположенных подобно элементам матрицы.

Первые два аргумента функции subplot задают количество рядов и колонок, на которые разделяется графическое окно. Третий аргумент — это порядковый номер подобласти, в которую следующая за subplot графическая функция должна поместить результаты своей работы. Например, запись

```
subplot(2, 2, 1); ezplot( 'x^2', [-8 8] );
subplot(2, 2, 2); ezpolar( 'sin(4*t)' );
subplot(2, 2, 3); ezmesh( 'x^2-y^2' );
subplot(2, 2, 4); x = [0:0.5:2*pi];
y=sin(x); plot(x, y, 'r--*');
```

Дает результат, представленный на рис.1.7.3.

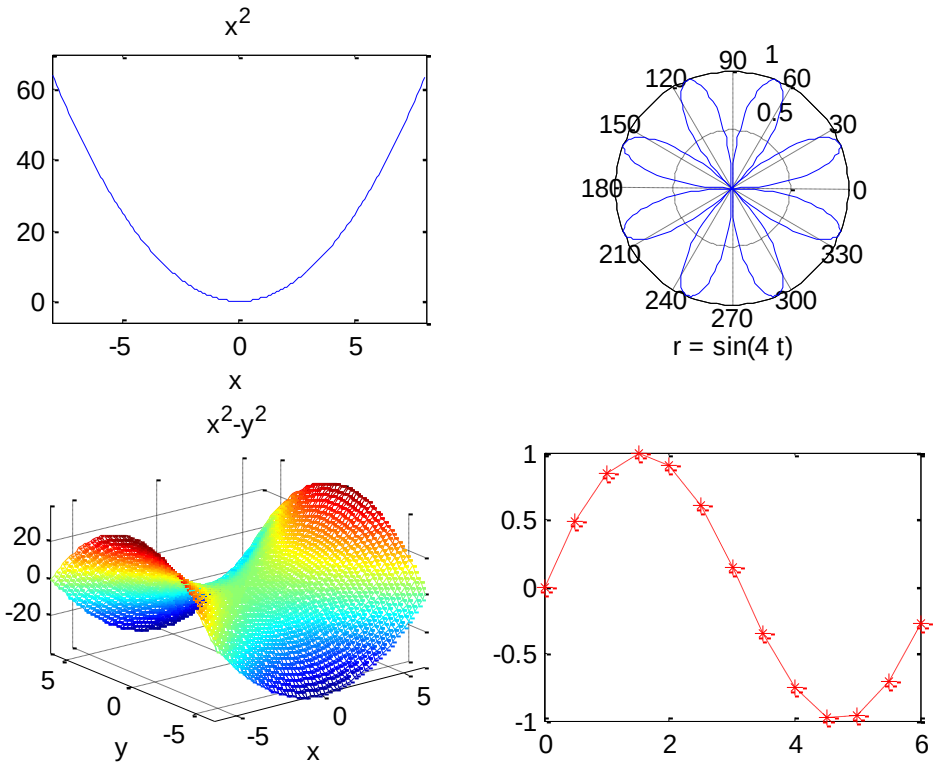


Рис.1.7.3 Вывод нескольких графиков, имеющих разную природу в подобластях графического окна

1.8. Изменение цвета графиков, которые строятся циклом

Если несколько функций строятся в цикле и заранее неизвестно, сколько графиков построит программа, но необходимо различать эти графики по цвету, полезно научиться изменять цвет графиков , которые строятся командой plot в цикле построения одной фигуры, задавая цвета в виде RGB –формата.

Можно задавать цвет графика в Матлаб в RGB формате:

```
plot([1,2],[2,3], 'color', [100]);
```

где [1 0 0] это цвет в RGB формате. Первое число - Красный, второе - Зелёный, третье - Синий. Значения меняются от 0 до 1. Таким образом вышеприведенная строчка команды нарисует красную линию. Если поставить все значения на 0 ([0 0 0]), то цвет будет чёрный, а если всё на 1, то цвет будет белый. Ну и комбинируя различные значения можно получить любой цвет.

Вот например фиолетовый будет:

```
plot([1,2],[2,3], 'color', [0.7500.75], 'LineWidth', 3);
```

Использование векторов цвета в формате RGB позволяет отобразить несколько графиков в одной фигуре Матлаб в том случае, когда заранее неизвестно, какое количество графиков придется отображать.

Например, организуем цикл расчета функций, число которых заранее не подсчитано и зависит от некоторого задаваемого параметра *n*:

n=2;% задаем параметр *n*

x = linspace(0,30);% определяем область значений переменой *x*

fori=1:n:10

y=sin(*x./i*); % в цикле рассчитываем функцию, которая зависит от параметра *n*

Color=[abs(sin(i)),abs(sin(10*i)),abs(sin(20*i))]; % задаем вектор цвета в формате RGB. Каждый график вновь рассчитываемой функции будет отображен на одной и той же фигуре, но другим цветом

plot(y,'color',Color);hold on;

end

Воспроизведите данную программу, но задайте свой параметр n и свою функцию

1.9 Анимация в MATLAB

В системе MATLAB нет процедур, позволяющих непосредственно создавать анимированные изображения, но есть средства, позволяющие проигрывать мультфильмы, кадрами которых являются заранее построенные графики.

Поэтому построение анимации в MATLAB осуществляется в два приема.

Сначала организуется цикл вида:

```
for k = 1:n
    Произвольный блок команд,
    обеспечивающих построение
    какого-либо графика.
    M(k) = getframe;
    end
```

где *n* – число кадров будущего мультфильма, а

M – массив, в который функцией getframe последовательно складываются кадры.

Затем собранный мультфильм проигрывается функцией movie, при вызове которой первым аргументом указывается массив кадров, вторым – количество повторных циклов анимации и третьим – скорость воспроизведения (число кадров в секунду).

Приведенные ниже фрагменты программного кода демонстрируют построение двумерной и трехмерной анимации и могут быть использованы вами в качестве шаблона для создания собственных мультфильмов.

Строим анимированный график на плоскости.

```
syms x
t = linspace( 0, pi, 300);
for k = 1:300
ezplot( sin(2*t(k))*sin(3*pi*x), [0 1] );
axis( [0 1 -1 1]);
grid on;
M(k) = getframe;
end
```

Проигрываем построенную анимацию три раза.

```
 movie(M, 3)
```

Строим анимацию в пространстве.

```
[x, y] = meshgrid( [0:0.01:1] );
t = linspace( 0, pi, 100);
```

```
for k =1:100
z = sin(2*pi*x).*sin(3*pi*y).*sin(2*t(k));
surf(x,y,z);
shading flat;
colormap(copper);
axis( [0 1 0 1 -1 1]);
M(k) = getframe;
        end
```

Проигрываем полученный мультфильм два раза со скоростью 30 кадров в секунду.

```
movie(M,1,30)
```

1.10. Считывание координат точек из графического окна

Часто, например, при нахождении нулевого приближения корня уравнения, бывает необходимо получить координаты одной или нескольких точек из активного графического окна.

В этом случае следует воспользоваться функцией ginput, которая имеет следующий синтаксис:

```
[x,y] = ginput(n)
```

Единственный и при этом необязательный аргумент *n* задает количество точек, которые требуется считать из графического окна. Два выходных параметра представляют собой векторы, в которые записываются абсциссы и ординаты считанных точек.

Вызов функции ginput приводит к появлению в текущем графическом окне управляемого мышью перекрестия. Щелчок левой клавишей мыши в любой точке графического окна приводит к добавлению ее координат в соответствующие массивы. После того, как указанное в аргументе функции ginput число команд будет считано, перекрестие автоматически пропадает.

Если вызвать функцию ginput без указания числа точек, то считывание координат будет продолжаться до тех пор, пока вы не нажмете клавишу Enter.

Найдем, например, приближенные значения трех ближайших к началу координат корней уравнения sin x =cosx.

```
ezplot('sin(x)')
holdon
ezplot('cos(x)')
[x ,y] = ginput(3)
```

x =

-3.6917

-1.7518

2.5046

y =

0.5149

-0.1908

0.6445

Часть II. Построение графического интерфейса пользователя (GUI)

2.1. Назначение графического интерфейса пользователя (GUI)

При разработке прикладных программ представляется полезным создание графического интерфейса пользователя[4]. Фактически, это создание среды расчета задач определенного класса без программирования со стороны пользователя. Как правило, такие интерфейсы имеет смысл разрабатывать для задач с несколькими параметрами, если предполагается неоднократное решение подобных задач. В таком случае целесообразно разработать графический интерфейс, который помогает пользователю получать результаты решения задачи (как правило, в графическом виде) при определенном выборе параметров. Такой интерфейс может быть также удобен при создании учебных задач, потому что обучающийся в таком случае основное внимание тратит не на программирование или решение задачи, а на подбор требуемых параметров, анализ и осмысление получающихся результатов.

Обязательными элементами графического интерфейса при решении научных и/или учебных задач должны быть[5]:

1. Одно или несколько окон для вывода графических результатов расчета.

2. Несколько редактируемых окон, с помощью которых задаются и/или изменяются значения параметров задачи.

3. Управляющие кнопки, которые позволяют запускать и останавливать процесс расчета, перерисовывать результаты, выходить из задачи.

4. Поясняющие надписи (статический текст).

Конечно, возможны и другие элементы управления, такие как прокручиваемые списки, радио-кнопки для выбора одного из многих вариантов и т.д., но сейчас мы подробно сосредоточимся только на перечисленных в списке четырех типах.

В системе MATLAB предусмотрены два способа организации интерфейса с пользователем[6]. Первый из них условно назовем динамическим. Он заключается в том, что на стадии выполнения программы создаются те или иные графические объекты и их свойствам присваиваются соответствующие значения. Для этого используются специальные функции, такие как Uicontrol, Uimenu и Uicontextmenu. Но для упрощения работы и создания однотипных элементов интерфейса в системе MATLAB имеется

специальная программа, которая позволяет на уровне визуального программирования, почти без написания кода создать требуемые элементы.

Вплоть до версии 2019 Матлаб имел встроенную среду GUIDE, которая помогала пользователю использовать полуавтоматический способ создания графического интерфейса с использованием технологий DAD ('Drag & Drop'). Эта технология подразумевает возможность не описывать каждый графический объект, помещаемый на интерфейс (форму), а использовать уже готовые элементы, собранные в одной библиотеке. Кроме того, используются CASE-технологии, которые позволяют получить программный код, описывающий элементы управления, помещаемые на форму, автоматически. (Т.е. мы только 'таскаем' элементы, а код генерируется автоматически). Это позволяет пользователю избежать большого объема рутинной работы, и сосредоточить свое внимание на решении задачи.

В настоящее время в версиях Матлаб 2020 и выше инструмент GUIDE отсутствует, разработчики предлагают вместо него пользоваться инструментом AppDesigner, позволяющим также создавать сетевые приложения. Однако инструмент AppDesigner недостаточно развит для того, чтобы в нем создавать полноценные графические приложения. Поэтому в новейших версиях Матлаб доступно редактирование, запуск и экспорт в файл Матлаб или AppDesigner ранее созданного GUIDE- приложения[7].

2.2. Разработка интерфейсных программ с помощью редактора GUIDE

Для вызова визуального редактора необходимо в командном окне MATLAB набрать команду guide или выбрать команду меню File|New|GUI. На экране дисплея будет выведено диалоговое окно GUIDE Quick Start вида, изображенного на рис.2.2.1.

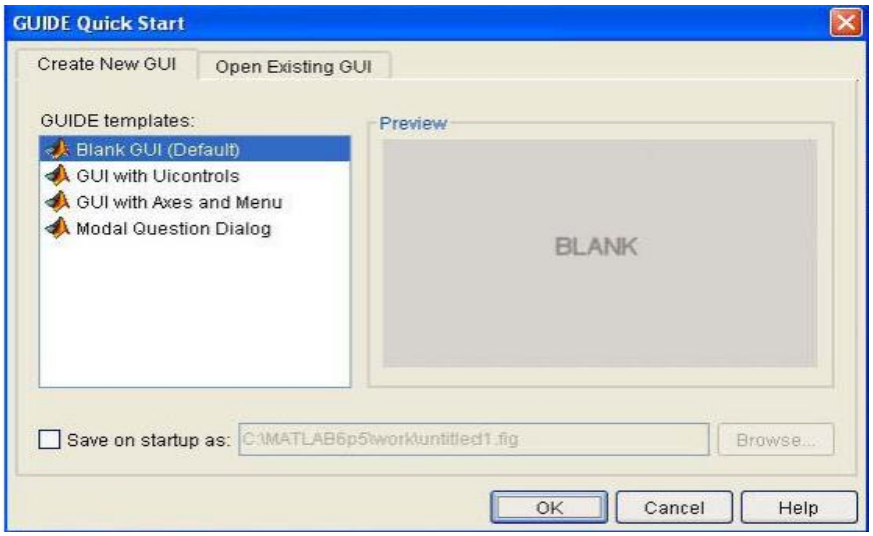


Рис. 2.2.1. Диалоговое окно GUIDE Quick Start

Это окно позволяет выбрать дальнейшее направление работ: создать новую заготовку для GUI-приложения (пустую или предопределенного формата) или открыть уже существующий (но требующий доработки или переработки).

Для создания "чистой" новой заготовки здесь нужно выбрать позицию Blank GUI и нажать OK. Тогда появится окно редактора– рис. 2.2.2, содержащее как саму заготовку, так и палитру графических элементов управления.

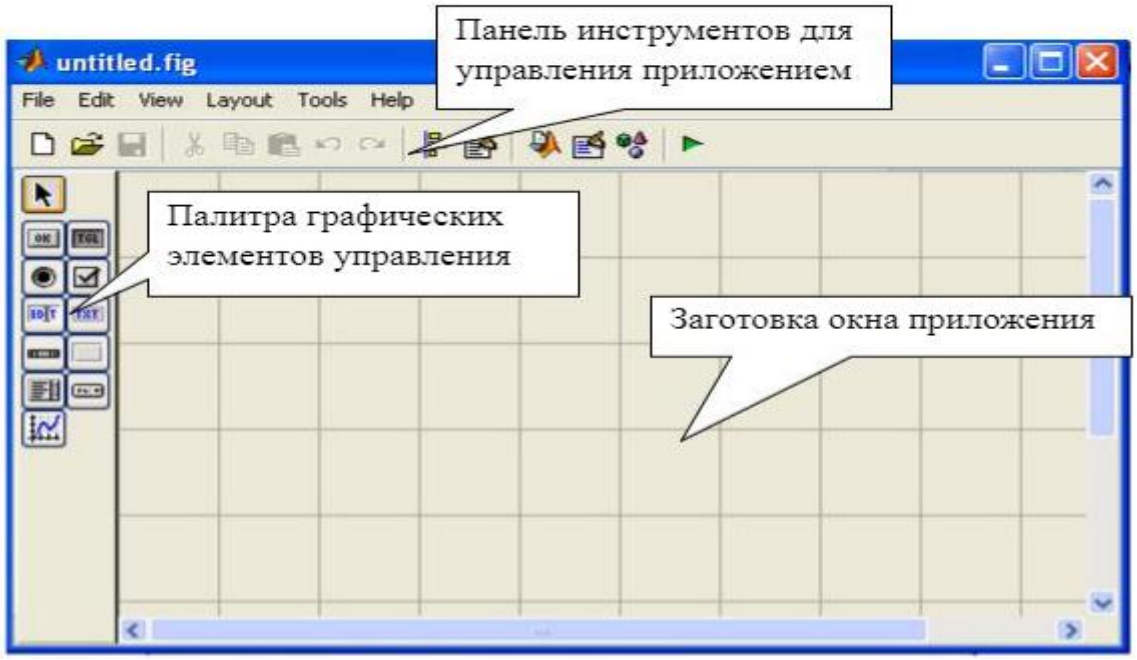


Рис. 2.2.2. Окно редактора GUIDE

На рис.2.2.3 в увеличенном виде представлена палитра элементов управления редактора GUIDE, при этом дана краткая характеристика каждого элемента.

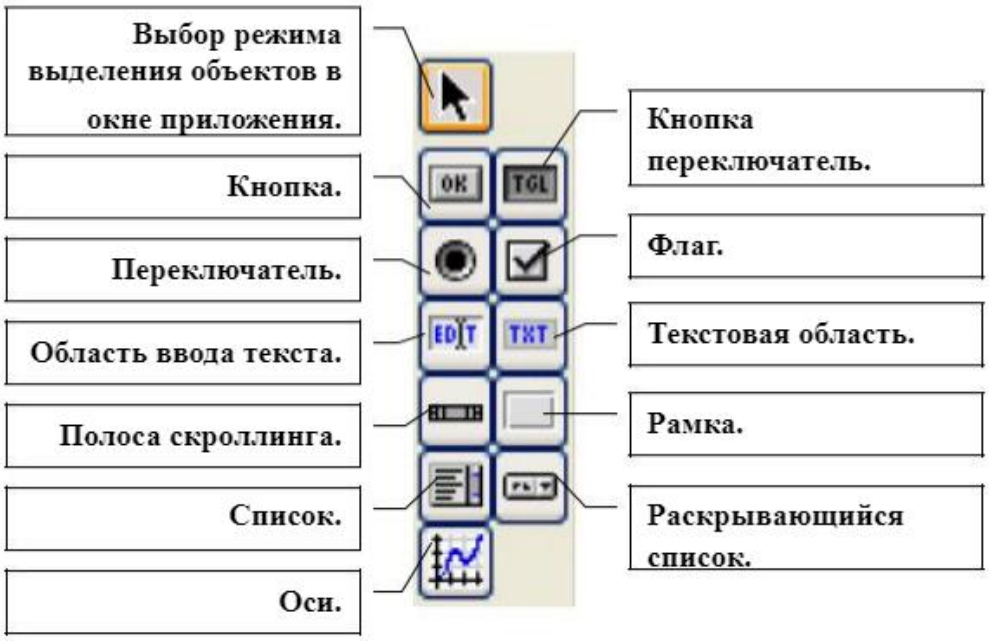


Рис. 2.2.3. Палитра элементов управления редактора GUIDE

Перед созданием графического интерфейса желательно "разработать проект" того, что вы хотите иметь в качестве интерфейса. В качестве демонстрационного примера мы рассмотрим разработку приложения для решения квадратного уравнения $ax^2 + bx + c = 0$. Для этого предположим, что нам хотелось бы видеть на форме главного приложения три окна: для ввода коэффициентов уравнения, два окна для отображения значений найденных корней и кнопку для запуска алгоритма вычисления корней после вода очередных значений коэффициентов. В этом приложении очень полезным бы окно, воспроизводящее график функции $y = ax^2 + bx + c$ и визуально подтверждающее расположение найденных корней. Эскиз окна предполагаемого приложения на рис.2.2.4.

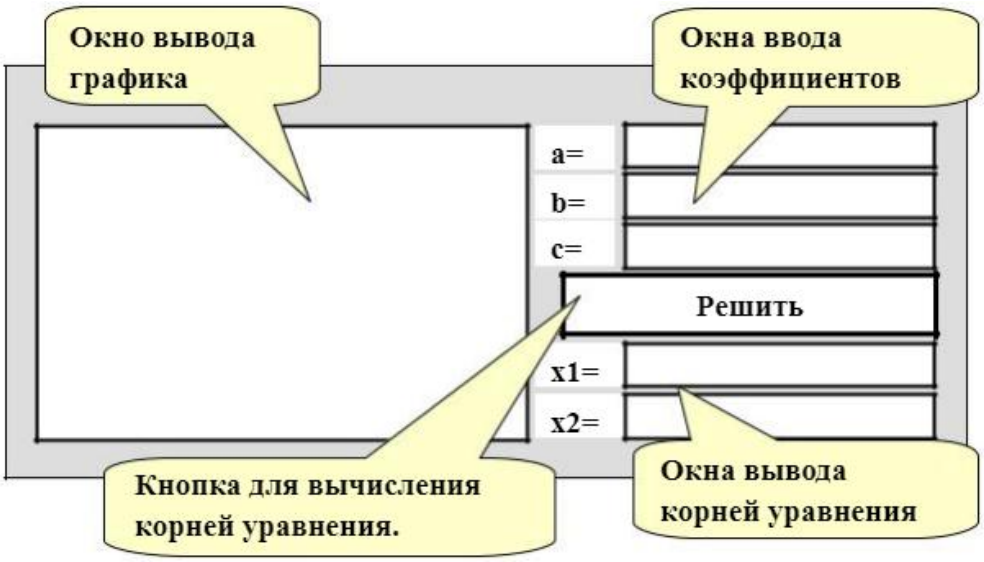


Рис.2.2.4. Эскиз окна предполагаемого приложения

На первом этапе необходимо разработать внешний вид интерфейса. Пусть по нашему предварительному эскизу интерфейс должен выглядеть приблизительно как на рис. 2.2.4. Элементами этого интерфейса являются окно вывода графика (**Axes/Оси** в терминах палитры элементов), пять статических надписей “a=”, “b=”, “c=”, “x1=”, “x2=” (**Static Text/Текстовая надпись**), три окна ввода/редактирования данных (**Edit Text/Область ввода текста**), два окна вывода данных (**Static Text/Текстовая надпись**) и одна кнопка (**Push Button/Кнопка**).

Теперь мы можем непосредственно приступить к созданию интерфейса пользователя. Для создания окна, в котором будет выводиться график, используется кнопка  (**Axes**). Щелкнув по этому элементу на панели палитры элементов управления и переведя мышь на панель рисунка, необходимо поместить крест, который будет на кончике мыши, в то

место, где должен находиться левый верхний угол окна. Нажав и удерживая левую кнопку мыши, необходимо вытянуть получающийся прямоугольник до нужных размеров.

Для создания редактируемых окон ввода используется кнопка  (**Edit Text**). Используется она так же, как при создании окна с осями. Сначала появляется мышь, нагруженная крестиком, с помощью которой строится прямоугольник ввода.

Надписи на панели рисунка, а также результирующих значений корней уравнения создаются с помощью кнопки  (**Static Text**), которая переносится и выравнивается аналогично выше описанному.

Для создания и размещения кнопки используется панель  с надписью **Push Button**. Способ размещения кнопки и выбора ее размера полностью совпадает с методом, описанным выше для окна редактирования и окна статического текста.

Построенные таким образом окна вывода и редактирования, области статического текста и кнопка, а также другие объекты можно выровнять и установить определенные промежутки между ними с помощью кнопки выравнивания  (**Align Objects**). Для задания ряда объектов, с которыми будут выполняться какие-либо действия (в частности выравнивание), необходимо их выделить, щелкая по каждому из них при нажатой клавише **Shift**. Выделенные объекты отмечаются черными точками вокруг соответствующих объектов. После этого необходимо на панели инструментов щелкнуть по кнопке  , после чего появится соответствующе окно выравнивания, с помощью которого несложно выполнить требуемые действия.

Размещение элементов управления на панели показано на рис. 2.2.5. При необходимости изменить размер какого-либо объекта (кнопки, окна и т.д.) необходимо щелкнуть по этому объекту с помощью левой кнопки мыши и с помощью мыши изменить требуемый размер так же, как и размер любого окна Windows. Выделенный объект можно перемещать, используя клавиатуру, с помощью клавиш вверх, вниз, влево, право, что очень удобно. Следует также отметить, что уже созданные и размещенные на панели рисунка объекты можно с помощью стандартных средств Windows копировать и вставлять, тем самым создавать новые объекты с такими же свойствами..

Выполняя вышеописанные действия, добиваясь внешнего сходства с разработанным нами эскизом, мы можем получить заготовку представленную на рисунке 5. После того как мы разместили требуемые объекты, мы можем задать их свойства. Например, задать соответствующее свойство (**String**) для надписей такие как “a=”, “b=”, и т.д. вместо

значений по умолчанию – “Static Text”, определить способ выравнивания текста в окнах ввода (свойство **HorizontalAlignment** значение **“left”**) и т.п. Для этого используется редактор свойств, который вызывается либо при помощи кнопки

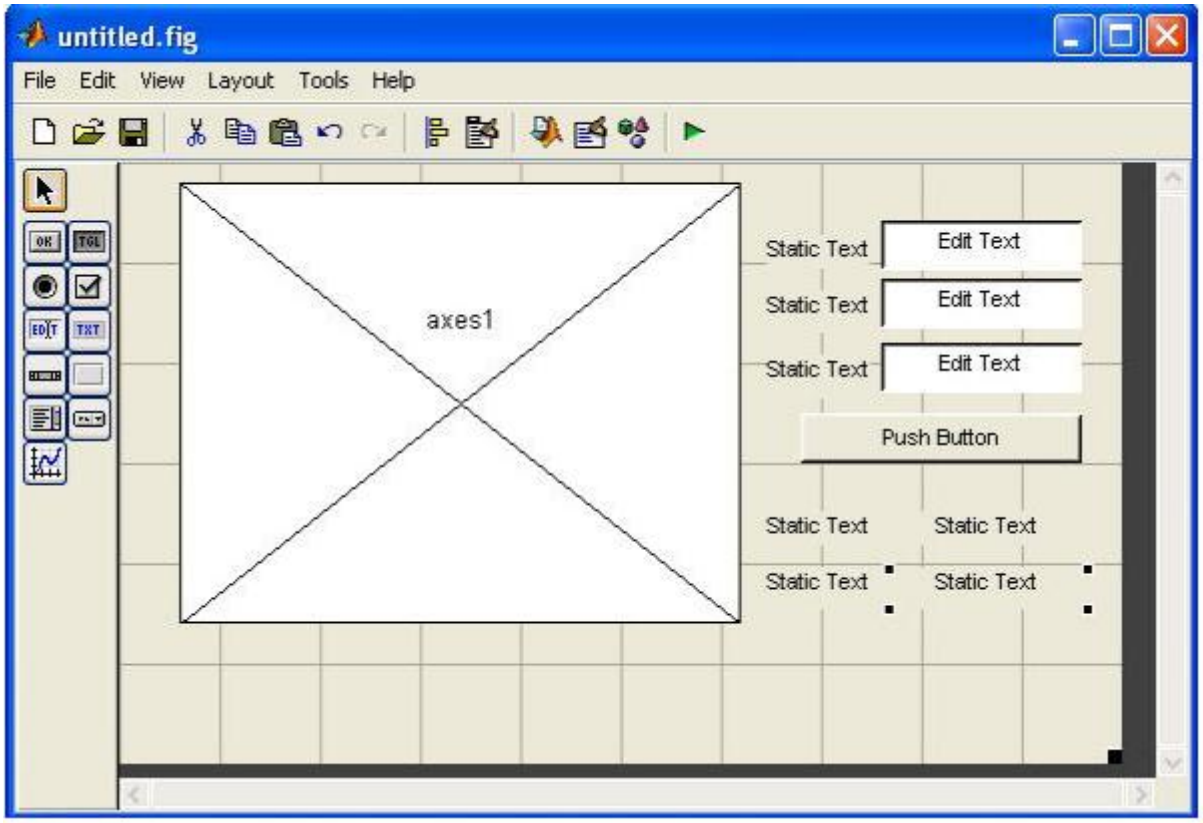


Рис.2.2.5. Размещение элементов управления на панели рисунка

Property Inspector  , либо с помощью двойного нажатия левой кнопкой мыши на соответствующем объекте на панели рисунка.

Например, чтобы изменить надпись на кнопке, мы двойным щелчком мыши откроем редактор свойств, показанный на рис.2.2.5, после чего изменим значение свойства **String** – “Push Button” на значение “Решить”.

Таким способом можно изменить все свойства выделенного объекта: видимость, цвет, расположение и его размеры, имя шрифта надписи, размер шрифта, выравнивание надписи и т.д. Всего для всех графических элементов управления предусмотрено 41 свойство, большинство из которых имеют вполне приемлемые значения по умолчанию.

Одно из полезных свойств графических объектов это свойство **Tag**. Оно содержит имя объекта (не путать с надписью на объекте – свойство **String**) и используется для доступа к конкретному объекту. По умолчанию, редактор GUIDE задает свойству **Tag** значения: text1, text2, text3, edit1, edit2, edit3, pushbutton1 и т.д. Пример задания показан на рис.2.2.6. Однако нам следует задать более осмысленные значения. Так

для информационных текстовых надписей свойству Tag зададим значения: textA, textB, textC, textX1, textX2; для текстовых надписей в которые мы будем выводить значения корней уравнения: textOutX1, textOutX2; для редактируемых окон ввода: edA, edB, edC; для кнопки pbtnSolve.

Рис. 2.2.6. Редактор свойств.

Выполнив настройку всех свойств объектов, необходимо сохранить наш проект. Для этого необходимо выполнить команду главного меню **File|Save As…** и задать имя файла, под которым результаты работы будут сохраняться. Назовем его, например, «MyRoot2». При этом создаются сразу два файла с одинаковым (заданным пользователем – «MyRoot2») именем и с расширениями fig и *m*. Бинарный файл с расширением fig содержит в закодированном виде сведения о графических элементах управления, а текстовой *m*-файл содержит код на *M*-языке.

Содержание последнего *m*-файла может обескуражить неподготовленного пользователя, но пугаться не стоит. Большая часть строк этого файла представляет подробный комментарий. Первая строка – это заголовок нашей функции “MyRoot2”, вызов которой из командной строки собственно и открывает созданное нами окно пользователя. Строки 2-25 представляют собой подробный комментарий нашей функции. В строках 26-46 располагаются команды инициализации приложения, и менять в этих строках ничего нельзя. Строки 47-65 занимает так называемая «открывающая» функция с именем MyRoot2_OpeningFcn, которая всегда вызывается при открытии приложения. Следующая

функция (строки 66-75) MyRoot2_OutputFcn условно можно назвать «закрывающей». Она срабатывает перед завершением приложения. Далее идут так называемые функции обработки событий (о которых пойдет речь дальше). К счастью, детально разбираться во всех тонкостях вышеперечисленных функций нам не придется, в чем вы убедитесь далее.

Программирование событий. Обработчик Callback.

Чтобы запустить наше приложение достаточно ввести в командной строке MyRoot2, или выполнить команду Tools|Run главного меню редактора GUIDE, или просто нажать кнопку  на панели инструментов. Приложение запускается в отдельном окне с заголовком MyRoot2 – рис.2.2.7.

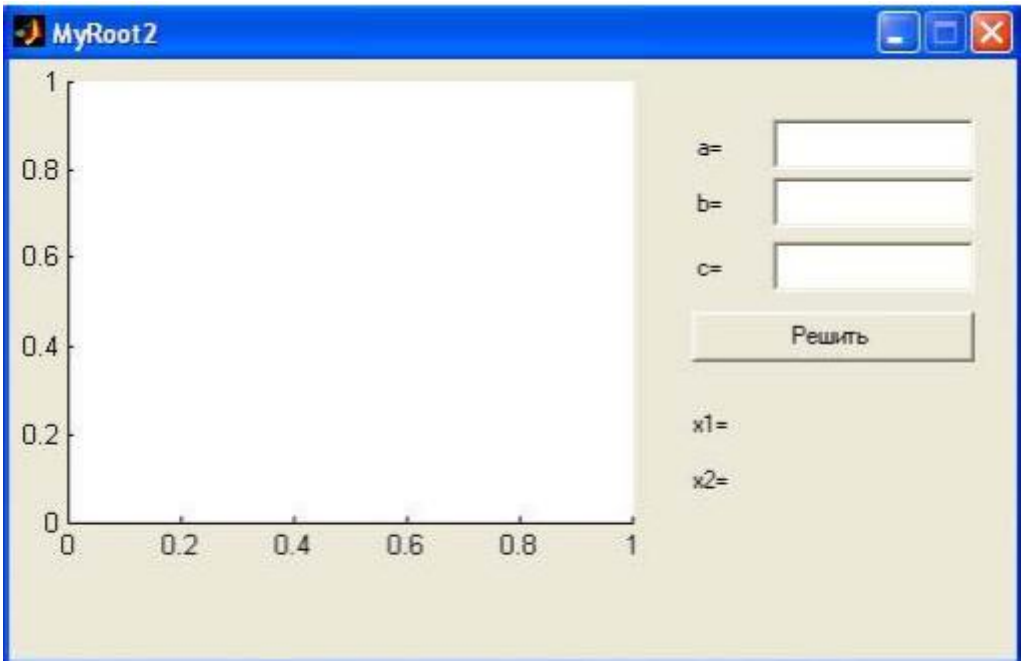


Рис.2.2.7. Приложение MyRoot2

Пользователь может нажимать на кнопку, вводить какую либо текстовую информацию в редактируемые окна, однако, при этом ничего полезного не происходит. Недостаточно разместить элементы интерфейса в окне приложения, следует позаботиться о том, чтобы каждый элемент выполнял нужные функции при обращении к нему пользователя. Так, при нажатии на кнопку необходимо чтобы был выполнен расчет корней уравнения и был выведен результат. Для этого необходимо запрограммировать обработчик событий Callback для кнопки pbtnSolve. Закроем наше приложение MyRoot2 и в редакторе GUIDE щелчком мыши выделим нашу кнопку pbtnSolve (с надписью «Решить»), после чего выполним команду View | Component Callbacks | Callback (Вид | Отобразить Callback-функции | Callback) – рис.2.2.8.

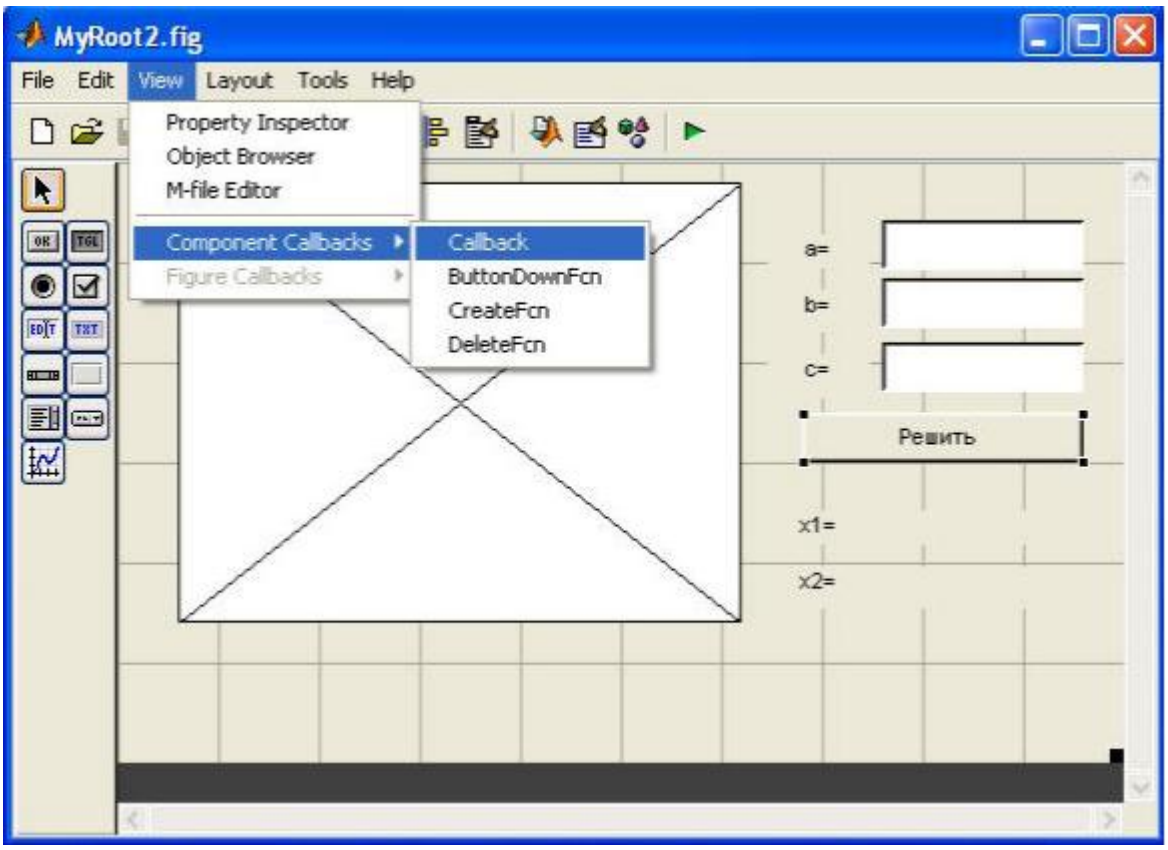


Рис. 2.2.8. Переход в тело Callback-функции

Конструктор интерфейса формирует заготовку для обработки указанного события и переводит нас в окно редактора кода на позицию, которую пользователь должен заполнить:

```
% --- Executes on button press in pbtnSolve.
function pbtnSolve_Callback(hObject, eventdata, handles)
% hObject    handle to pbtnSolve (see GCBO)                    of
% eventdata  reserved - to be defined in a future version MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

Комментарий в первой строке свидетельствует о том, что функция выполняется при щелчке по кнопке pbtnSolve. Среди параметров, передаваемых функции, важны первый и третий. Первый **hObject** является указателем (дескриптором) на объект, инициировавший событие callback, т. е. на кнопку pbtnSolve.

Третий **handles** — структура, содержащая поля с указателями на все интерфейсные элементы, установленные на фигуре (в нашем примере таковыми являются указатели handles.edA на редактируемое окно edA, handles.edB на редактируемое окно edB, handles.textOutX1 на текстовую область textOutX1 и т.д.).

Теперь приступим непосредственно к заданию алгоритма действий, который будет выполняться при нажатии на кнопку pbtnSolve («Решить»).

```
% --- Executes on button press in pbtnSolve.
```

```
function pbtnSolve_Callback(hObject, eventdata, handles)
% hObject    handle to pbtnSolve (see GCBO)
% eventdata  reserved - to be defined in a future version MATLAB
% handles    structure with handles and user data (see GUIDATA)

%сохраняем дескрипторы
hAxes=handles.axes1;
hA=handles.edA;         %на окно ввода edA
hB=handles.edB;         %на окно ввода edB
hC=handles.edC;         %на окно ввода edC
hX1=handles.textOutX1; %на окно вывода textOutX1 hX2=handles.textOutX2; %на окно вывода textOutX2
%делаем активным оси с дескриптором hAxes и очищаем их axes(hAxes);
cla;

%Получаем из редактируемых окон ввода edA,edB,edC %их значения с помощью функции get и с помощью %функции str2num переводим их в числовой вид str=get(hA,'String');
a=str2num(str);        str=get(hB,'String');        b=str2num(str);
str=get(hC,'String'); c=str2num(str);
%формируем вектор коэффициентов полинома - p
%и с помощью функции roots находим корни уравнения p=[a b c];
x=roots(p)

%в переменные xr1 и xr2 помещаем действительную %часть корней уравнения
xr1=real(x(1));
xr2=real(x(2));

%задаем промежуток [xmin,xmax] для построения %графика функции y=ax^2+bx+c xmin=min(xr1,xr2)-0.5;
xmax=max(xr1,xr2)+0.5; %формируем вектор аргументов xx=xmin:0.1:xmax;
%вычисляем вектор значений и строим график yy=polyval(p,xx);
plot(xx,yy) grid on
%выводим значения найденных корней в
%текстовые области textOutX1 и textOutX2 %предварительно преобразуя числовые значения %корней x(1) и x(2) в строковые с помощью
%функции num2str set(hX1,'String',num2str(x(1)));
set(hX2,'String',num2str(x(2)));
```

В тексте программы приведен подробный комментарий, поясняющий каждое действие. Отметим только новые для нас функции, это:

cla (Clear Axes) – удаляет все графики в текущих осях;

•**str2num(str)** – преобразует строку **str** в число (если возможно, иначе будет ошибка);

•**num2str(x)** – преобразует **x** число в строку;

Если мы теперь запустим наше приложение на исполнение, то получим работоспособную интерфейсную программу, показанную на рис.2.2.9.

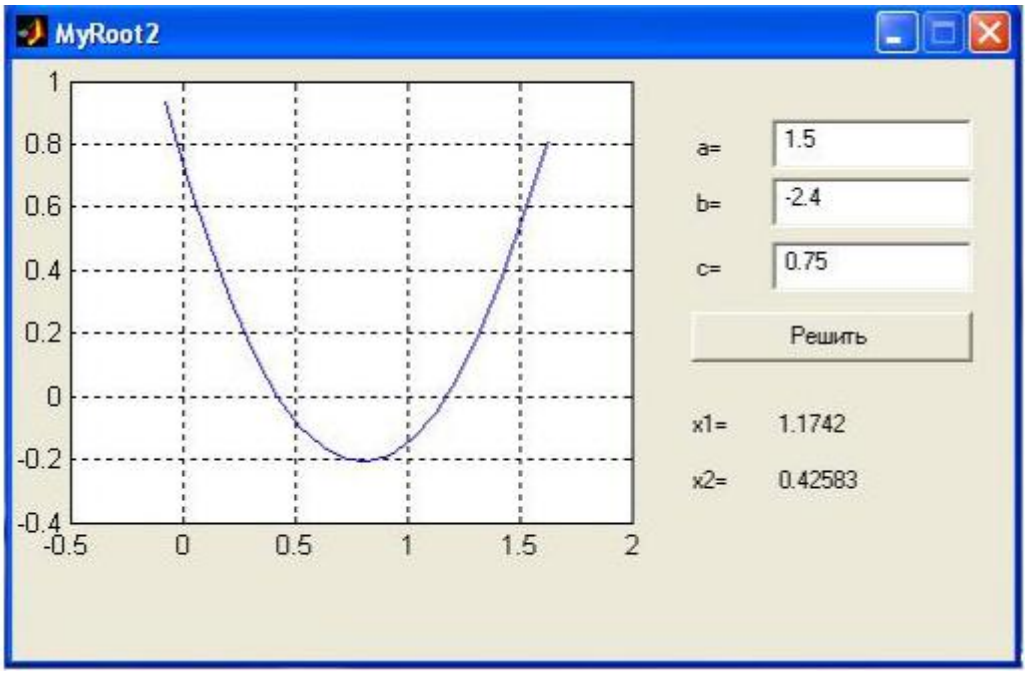


Рис. 2.2.9. Программа решения квадратного уравнения

2.3. Построение графического интерфейса пользователя «вручную»

Работая в интерактивном режиме, мы имели дело с графическими окнами (команда figure), в которых сама система MATLAB создавала графический объект Axes, на поверхности которого осуществлялась прорисовка осей системы координат и изображение графиков функций (в том числе трёхмерных).

Для подготовки информации, необходимой для построения графиков функций, мы вводили команды в командном окне MATLABa, получая необходимую числовую информацию и накапливая её в переменных рабочего пространства.

Если требуются громоздкие и сложные предварительные вычисления, то такую работу лучше всего оформить в виде отдельных *M*-функций. Вызывая эти функции из командного окна MATLABa, получаем необходимую информацию и запоминаем её в переменных-массивах системы MATLAB. Затем по команде figure создаём новое графическое окно, после чего командами plot, plot3, mesh, surf или surfl воспроизводим графики.

Такая работа требует больших знаний команд и функций системы MATLAB, знаний основ программирования в рамках этой системы. Поэтому повторить все эти действия

стороннему пользователю будет затруднительно. Даже самому разработчику для выполнения многочисленных повторяющихся заданий было бы удобнее иметь некоторый объединяющий механизм, позволяющий удобным и наглядным способом выполнять все виды разнородных работ.

Для этих целей в системе MATLAB применяются графические элементы управления, располагающиеся на поверхности графических окон и позволяющие вводить и читать числовую и текстовую информацию, нажатием кнопок инициировать выполнение нескольких М-функций сразу с последующим автоматическим показом результатов вычислений на поверхности графического объекта Axes того же графического окна. Из командного окна MATLABa потребуется всего лишь один раз вызвать М-функцию, создающую такое графическое окно.

Мы будем рассматривать следующие графические объекты управления: объект Axes, предназначенный для построения графиков функций, а также несколько объектов общего типа uicontrol. К последним относятся *командные кнопки*, *текстовые поля* с возможностью редактирования текста и без такой возможности, *переключатели* и *списки* .

Помимо графических элементов управления в графических окнах можно создавать также дополнительные команды меню (помимо стандартных команд, присутствующих во всех без исключения графических окнах), но мы здесь этой возможностью пользоваться не будем.

Все перечисленные выше графические объекты выполнены в системе MATLAB по *объектной технологии* и характеризуются присущим им *набором свойств*. Меняя последние, мы модифицируем их внешний вид и поведение. Чтобы изменить свойства графических объектов, нужно получить доступ к ним по их *описателю* (число, уникально идентифицирующее конкретный объект).

Описатель графического объекта возвращают функции, создающие эти объекты. Запомнив эти описатели в глобальных переменных, мы всегда впоследствии будем иметь возможность доступа к ним. Другим, более простым способом получения описателей, является использование специальных функций MATLABa, таких, например, как функция gcf (Get Current Figure - получить описатель текущего графического окна).

Например, команда figure создаёт графическое окно стандартного (заданного по умолчанию) цвета. Если нам требуется изменить цвет на красный, то это легко выполнить, получив описатель окна функцией gcf, а затем присвоив новое значение *свойству* 'Color' (цвет):

hWnd = gcf; set(hWnd, 'Color', 'red');

или ещё короче

set(gcf, 'Color', 'red')

Все графические объекты (они сами являются окнами в смысле операционной системы Windows) на поверхности графического окна создаются по иерархической схеме "родитель – потомство". Родительским окном для элементов управления служит само графическое окно MATLABa. С учётом введённого понятия становится удобным применение для поиска описателей графических объектов функции findobj:

hArray = findobj(hParent, *'имя_свойства', значение_свойства*)

Эта функция отыскивает все объекты, являющиеся потомками объекта с описателем hParent и имеющие для свойства *'имя_свойства'* значение, указанное в параметре функции *значение_свойства*. Она возвращает их описатели в массиве hArray. Этой функцией широко пользуются для доступа к описателям конкретных элементов управления, которые являются потомками графического окна MATLABa, так что вместо hParent можно использовать gcf. В качестве удобного для поиска свойства часто используется свойство 'Tag' (*ярлык*), значением которого является произвольно присваиваемый объекту (элементу управления) текстовый идентификатор. Если мы разным объектам присвоим разные идентификаторы в качестве значений свойства 'Tag', то не будет проблем с поиском описателей этих объектов.

А теперь создадим для примера командную кнопку на поверхности графического окна:

```
hF1 = figure;
 uicontrol( hF1, 'Style','pushbutton', 'String', 'MyButton1',…
                                                'Position', [ 10 10 70 30 ] );
```

Любой элемент управления создаётся функцией uicontrol (имя функции, создающей объект, всегда совпадает с именем этого объекта.Такие функции в программировании принято называть *конструкторами*), у которой первым параметром идёт описатель родительского окна, а затем по очереди перечисляются имена и значения свойств, которым мы явно придаём собственные значения (а остальные, менее важные для нас свойства получают значения по умолчанию). В итоге получается следующее графическое окно на рис.2.3.1, в котором явственно видна кнопка. Эта кнопка визуально действует безупречно – с помощью левой клавиши мыши она нажимается (виден процесс заглубления поверхности кнопки) и отжимается, но при этом не происходит никаких действий в качестве последствий нажатия. Это происходит потому, что мы ещё не приписали этой кнопке никаких функций, выполнение которых должно быть реакцией на нажатие.

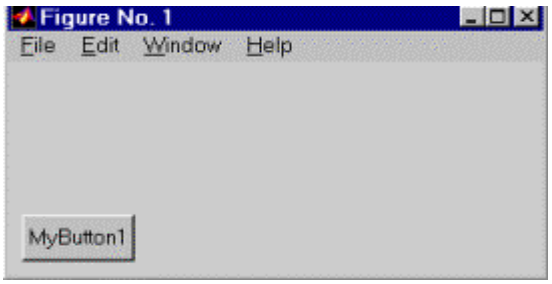


Рис. 2.3.1 Командная кнопка, созданная на поверхности графического окна

В функции uicontrol, создающей элемент управления, самым важным параметром после описателя родительского окна является свойство 'Style', так как оно задаёт тип управляющего элемента. Задав для этого свойства значение 'pushbutton', мы создали именно кнопку (а не редактируемое поле, или что-нибудь ещё).

Имена двух других свойств говорят сами за себя: String задаёт надпись на поверхности кнопки (в данном случае это MyButton1), а Position имеет значением вектор-строку из четырёх чисел и задаёт положение управляющего элемента относительно левого нижнего угла графического окна. < /P >

Для ввода и редактирования входной информации (числовой и текстовой) предназначены *текстовые поля с возможностью редактирования*. Они создаются командой

uicontrol(hParent, 'Style', 'edit', …),

где многоточие означает дополнительные свойства, одно из которых – 'Position', нужно указывать практически обязательно, иначе управляющий элемент будет расположен в неудачном месте. Теперь, если в дополнение к ранее созданной кнопке, выполнить команду

```
uicontrol( hF1, 'Style', 'edit', 'Position', [ 100 10 70 30 ],…
                                    'BackgroundColor', 'white',…
                                    'HorizontalAlignment', 'left' );
```

то графическое окно примет следующий вид, как на рис.2.3.2.

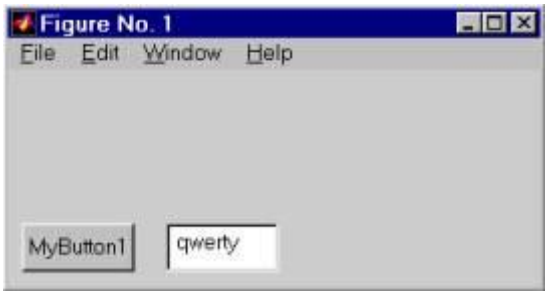


Рис. 2.3.2 Добавление на поверхность графического окна области редактируемого текста

Для редактирующего элемента мы прописали свойство 'BackgroundColor' (цвет фона - сделали его белым вместо тёмно-серого по умолчанию) и свойство 'HorizontalAlignment' (тип выравнивания текста в горизонтальном направлении - сделали "*выравнивание по левой границе*" вместо "*выравнивание по центру*" , имеющему место по умолчанию).

Заметим здесь, что свойство 'BackgroundColor' *нельзя произвольно задавать для кнопок* (требование операционной системы Windows).

Как и в случае с кнопкой созданный редактирующий элемент сразу же проявляет свою функциональность: в нём можно мышью поставить тестовый курсор (переместить на него *фокус ввода с клавиатуры*) и вводить произвольный текст и числа с клавиатуры. На рисунке видно, что мы успели ввести "магическое буквосочетание" qwerty.

Теперь создадим *текстовое поле без редактирования* (часто называют статическим текстом). На поверхности этого элемента можно расположить произвольный поясняющий текст. Этот текст можно изменять программно, но его нельзя ввести с клавиатуры. Вот пример вызова функции uicontrol, создающий не редактируемое текстовое поле, которое расположено чуть выше редактирующего элемента:

```
uicontrol( hF1, 'Style', 'text', 'Position', [ 100 50 70 30 ],…
                    'BackgroundColor', 'white', 'String','Hello, World!',…
                    'HorizontalAlignment', 'center' );
```

Теперь наше графическое окно содержит уже три элемента управления – рис. 2.3.3.

Рис..2.3.3. Добавление на поверхность графического окна области надписи

Как видно из приведённого фрагмента, для создания не редактируемого элемента свойство 'Style' должно иметь значение 'text'.

Создадим теперь элемент управления, называемый "*флажком*" (**checkbox**):

```
uicontrol( hF1, 'Style', 'checkbox', 'Position',[10 100 150 30],…
                    'String','Important option','HorizontalAlignment', 'left' );
```

Графическое окно примет вид как на рис. 2.3.4.

Рис. 2.3.4. Добавление на поверхность графического окна области «флажка»

Этот элемент управления может пребывать в *одном из двух состояний*: *отмеченном* или *неотмеченном*. В первом из них в квадрате должна стоять "*галочка*" (отметка), которую проставляют щелчком левой кнопки мыши на этом элементе управления. Повторный щелчок убирает "галочку".

Наконец, создадим *список*, содержащий несколько имён функций на рис.2.3.5.

uicontrol(hF1, 'Style', 'listbox', 'Position',[200 10 100 80],…

'String',{'sin','cos','tan'},'HorizontalAlignment', 'left');

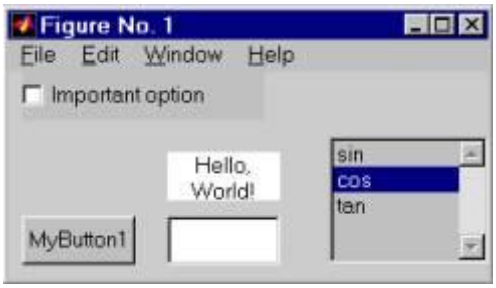


Рис. 2.3.5 Добавление на поверхность графического окна области списка

Так как список (свойство 'Style' должно быть 'listbox') содержит сразу несколько строковых значений для свойства 'String', а должно быть одно значение, то это единственное значение является *массивом ячеек* (каждый элемент этого массива есть текстовая строка, то есть массив типа char).

Внутри списка щелчком мыши можно *выделить* (подсветить) какую-либо из его строк. На рисунке показано, что выделена вторая строка.

С графическим объектом Axes мы знакомились, когда обсуждали вопрос о построении графиков функций. Теперь, когда мы в пределах графического окна системы MATLAB собираем весь набор управляющих и отображающих элементов, работа с объектом Axes по-прежнему является одной из наиболее важных.

Создаётся объект Axes с помощью функции axes (конструктор объекта). На нашем учебном окне осталось мало свободного места, однако мы ухитримся втиснуть туда

графический объект Axes, для чего специально уменьшим размер шрифта ('FontSize'), используемого для надписей на осях координат объекта Axes:

axes('Parent', hF1, 'Color', [1 1 1], …

'Units', 'points', 'Position', [12 33 40 22], …

'FontSize', 6);

В результате получается следующая картинка, не претендующая на оптимальность в размещении объектов в пределах графического окна – рис.2.3.6.

Рис. 12.6 Добавление на поверхность графического окна области координатных осей

Все команды, создающие графическое окно, объект Axes и элементы пользовательского интерфейса удобно поместить в М-файл. Этот файл будет служить сценарием создания такого окна, причём от пользователя потребуется минимальное количество действий - набрать в командном окне MATLABa имя М-файла и нажать клавишу Enter.

Приведём пример М-файла, создающего в графическом окне один объект Axes, один список, четыре редактируемых поля для ввода числовой информации и две командные кнопки, не считая нескольких текстовых полей (без возможности редактирования), содержащих подписи, пояснения и разграничительные линии. Вот содержимое этого файла:

```
global hFig1, hAxes, %----- the file Proj1Script.m ----------
global hEd1, hEd2, hEd3, hEd4
 global hList global hBut1, hBut2
global hTxt1,hTxt2,hTxt3,hTxt4,hTxt5,hTxt6,hTxt7
 %-----------------------------------------------
 hFig1 = figure
 hAxes = axes('Parent',hFig1,'Color', [ 1 1 1],'Units', ...
'points', 'Position', [ 20 60 230 230 ],'FontSize', 10 );
%-----------------------------------------------
```

```
hEd1 = uicontrol( hFig1,'Style','edit','BackgroundColor',[1 1 1], ...
                  'Position',[435 400 190 30],'HorizontalAlignment','left');
hEd2 = uicontrol( hFig1,'Style','edit','BackgroundColor',[1 1 1], ...
                   'Position',[435 320 190 30],'HorizontalAlignment','left');
hEd3 = uicontrol( hFig1,'Style','edit','BackgroundColor',[1 1 1], ...
                   'Position',[435 230 190 30],'HorizontalAlignment','left');
hEd4 = uicontrol( hFig1,'Style','edit','BackgroundColor',[1 1 1], ...
                   'Position',[435 150 190 30],'HorizontalAlignment','left')
hList = uicontrol( hFig1,'Style','listbox','Position',[485 15 140 90], ...
                  'String',{'sin','cos','MyFunct1','MyFunct2','f3' }, ...
                   'HorizontalAlignment', 'left' );
hBut1 = uicontrol( hFig1,'Style','pushbutton','Position',[60 25 140 40], ...
                    'String', 'GO' );
hBut2 = uicontrol( hFig1,'Style','pushbutton','Position',[240 25 140 40], ...
                   'String', 'CLEAR' )
 hTxt1 = uicontrol( hFig1,'Style','text','BackgroundColor',[0.7 0.7 0.7], ...
                  'Position', [90 475 240 20],'String','Euler solution' );
hTxt2 = uicontrol( hFig1,'Style','text','BackgroundColor',[0.7 0.7 0.7], ...
                  'Position', [435 440 60 20],'String','X initial' );
hTxt3 = uicontrol( hFig1,'Style','text','BackgroundColor',[0.7 0.7 0.7], ...
                  'Position', [435 360 60 20],'String','Y initial' );
hTxt4 = uicontrol( hFig1,'Style','text','BackgroundColor',[0.7 0.7 0.7], ...
                   'Position', [435 260 60 20],'String','X final' );
hTxt5 = uicontrol( hFig1,'Style','text','BackgroundColor',[0.7 0.7 0.7], ...
                  'Position', [435 180 60 20],'String','N' );
hTxt6 = uicontrol( hFig1,'Style','text','BackgroundColor',[0 0 0], ...
                  'Position', [430 5 1 900] );
hTxt7 = uicontrol( hFig1,'Style','text','BackgroundColor',[0.7 0.7 0.7], ...
                 'Position', [435 140 240 20],'String','Choose function' );
```

В результате получается вот такое графическое окно – рис.2.3.7.

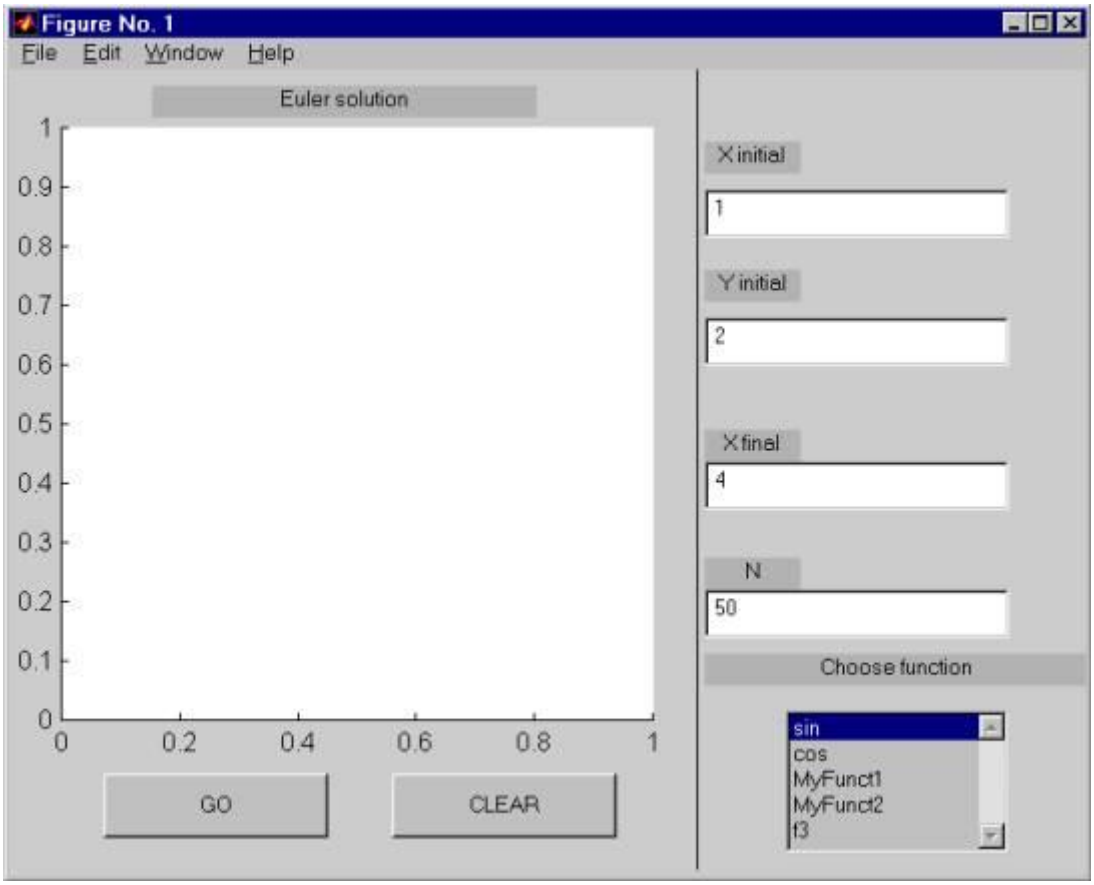


Рис. 2.3.7 Графический интерфейс пользователя для вычисления нескольких функций и построения их графиков

Это окно предназначено для численного решения дифференциального уравнения

y'(x) = F(x); Xinitial < x < X final; Yinitial = y(Xinitial);

где функция *F*(*x*) выбирается из списка, расположенного в правом нижнем углу графического окна. Остальные параметры задаются с клавиатуры в редактируемых полях. Кнопка GO предназначена для начала расчётов по методу Эйлера и показа результата в виде графика на поверхности объекта Axes. Кнопка CLEAR позволяет очистить объект Axes от содержимого.

Как мы видим из рис.2.3.7 в нем предусмотрены расчёты со стандартными функциями sin(x) и cos(x), а также с собственными функциями MyFunct1(x), MyFunct2(x) и f3(x). Последние должны быть записаны в M-файлы MyFunct1.m, MyFunct2.m, f3.m.

Число *N*, которое вводится перед расчётами с клавиатуры, определяет шаг интегрирования, который равен

(Xfinal - Xinitial) / N.

Чтобы запустить программы подсчётов, нужно не только записать алгоритм решения на М-языке и сохранить текст в М-файле, но и связать эти программы с командными кнопками.

Функции, которые связывают с графическими элементами управления, называют callback-функциями (их не вызывают наши М-функции; они вызываются средой MATLAB у нас "за спиной", отсюда и происхождение их названия).

Чтобы связать наши кнопки с callback-функциями, которые будут вызываться средой MATLAB при "нажатии" на эти кнопки, при создании кнопок функцией uicontrol нужно прописать свойство 'Callback', указав ему в качестве значения имя М-функции. Это значит, что в ранее представленном файле сценария Proj1Script.m нужно изменить строки, касающиеся создания кнопок. Вот этот изменённый код, позволяющий связать кнопки сих callback-функциями:

```
hBut1 = uicontrol( hFig1,'Style','pushbutton','Position', …
                [60 25 140 40],'String', 'GO','Callback','MyGO');
hBut2 = uicontrol( hFig1,'Style','pushbutton','Position', …
     [240 25 140 40],'String', 'CLEAR','Callback','MyCLEAR');
```

Если функция, которая будет вызываться при нажати на кнопку GO будет записана в файле MyGO.m, то в качестве значения свойства 'Callback' обязательно нужно написать значение в виде 'MyGO', и нельзя писать 'MyGO.m' (возникнет ошибочная ситуация).

Поправив файл Proj1Script.m, перейдём теперь к написанию функций MyGO и MyCLEAR. Начнём с более простой функции MyCLEAR. Вот текст этой функции:

```
function MyCLEAR
%----------
global hAxes
axes(hAxes);
cla;
```

Мы уже в файле Proj1Script.m обозначили все описатели созданных нами графических объектов как глобальные. Таким образом, у нас имеется очень простой к ним доступ. Все функции, где эти описатели маркированы как *глобальные*, имеют к ним непосредственный доступ. Именно так всё и происходит в функции MyCLEAR (её мы записываем в файл MyCLEAR.m). Поясним текст этой функции. Здесь команда cla стирает содержимое объекта Axes. Предварительный вызов функции

```
axes( hAxes )
```

позволяет гарантировать, что будет стёрто содержимое именно объекта Axes с описателем hAxes, так как такой вызов делает активным тот объект, описатель которого

указан в качестве параметра. У нас в данной конфигурации имеется всего один объект Axes, так что этот вызов не является обязательным, но он иллюстрирует решение проблемы в случае нескольких объектов Axes.

Теперь займёмся функцией MyGO, которая запускается по нажатию кнопки GO. Эта функция опрашивает редактируемые поля, в результате чего инициализируются необходимые для счёта переменные, после чего производит вычисления решения дифференциального уравнения в соответствии с простейшим алгоритмом Эйлера:

```
function MyGO
global hAxes hEd1 hEd2 hEd3 hEd4 hList
%-- get string information from editboxes -----
str1=get( hEd1,'String' ); str2=get( hEd2,'String' );
str3=get( hEd3,'String' ); str4=get( hEd4,'String' );
 %-- convert strings to numbers ----------
x0 = str2num( str1 ); y0 = str2num( str2 );
xf = str2num( str3 ); N = str2num( str4 );
%-- get function name ------------------------
index = get( hList, 'Value' );
cellArr = get( hList, 'String' );
funName = cellArr{ index };
%-- Euler algorithm ---------------------------
axes( hAxes );
dx = ( xf - x0 ) / N;
X=x0; Y=y0;
for k=1:N
    Y=[ Y, Y(end) + feval( funName, X(end) ) * dx ];
    X=[ X, X(end) + dx ];
End
%-- graph plotting ---------------------------
 plot( X , Y );
```

С помощью функций get, указывая им в качестве первого аргумента описатели редактирующих полей, мы получаем значения свойства 'String', то есть текстовое содержимое этих полей, которое вводится пользователем с клавиатуры. Это и есть входные данные. Только пока что числовые данные представлены в строковой (текстовой) форме.

Для получения имени функции, представляющей правую часть дифференциального уравнения, сначала у списка запрашиваем через свойство 'Value' индекс подсвеченной (выбранной или отселектированной) строки, затем читаем значение свойства 'String'. Последнее для списка является массивом ячеек, откуда и выбираем имя функции с помощью операции индексирования массива ячеек.

С помощью функций str2num преобразуем текстовые величины в числа. Затем устанавливаем начальные значения для массива X значений аргумента и массива Y значений функции на сетке. *Шаг сетки* (шаг интегрирования) dx вычисляется делением длины отрезка интегрирования на количество шагов N, которое пользователь вводит с клавиатуры. Наращивание массивов осуществляется операцией конкатенации, в то время как очередное значение функции вычисляется согласно алгоритму метода Эйлера и равно

feval(funName, X(end)) * dx

Ключевое слово системы MATLAB end означает ссылку на последний элемент массива, что нам и требуется в данном случае. С помощью функции feval осуществляется вызов функции с именем funName, прочитанным из списка, и ей передаётся аргумент X(end)(текущее значение независимого аргумента).

В конце концов осуществляется построение графика вычисленного решения дифференциального уравнения с помощью функции plot. Перед вызовом этой функции с помощью axes(hAxes) гарантируется, что наш объект типа Axes является активным, и именно в нём функция plot строит график функции.

Проверим работу приложения для следующих пользовательских функций MyFunct1, MyFunct2 и f3. Текст функции MyFunct1

```
function ret = MyFunct1( x )
 ret = x ^ 3;
```

записываем в файл MyFunct1.m. Текст функции MyFunct2

```
function ret = MyFunct2( x )
 ret = 1 / sin( x );
```

записываем в файл MyFunct2.m. Наконец, текст функции f3

```
 function ret = f3( x )
ret = sin( x * x );
```

записываем в файл f3.m. Начинаем с простой функции cos(*x*). Окно показано на рис.2.3.8.

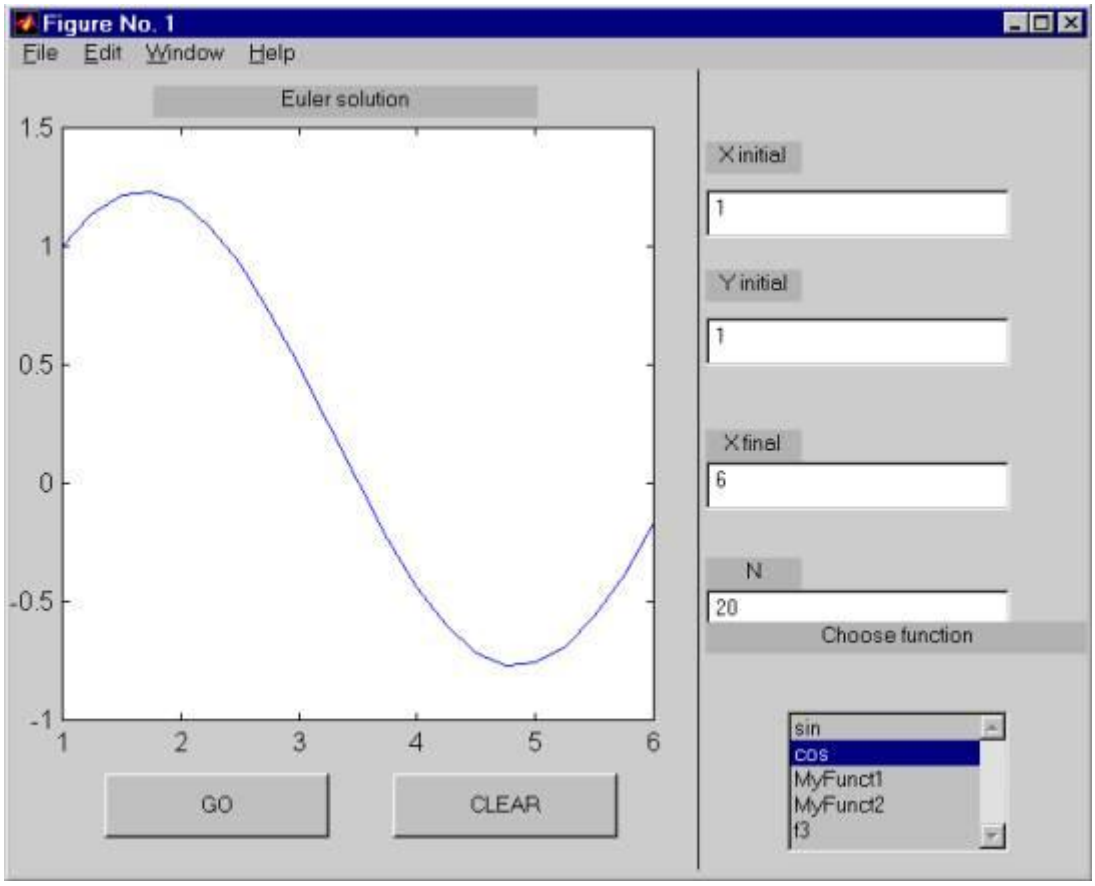


Рис.12.8 Графический интерфейс пользователя при вычислении функции cos(*x*)

Так как дифференциальное уравнение y'(x)=cos(x) имеет аналитическое решение

y(x) = Yinitial - sin(Xinitial) + sin(x)

то в полученном графике легко узнать график функции sin(x), сдвинутый вдоль вертикальной оси вверх.

Также очевидые решения получаются для функций sin и MyFunct1, так как и в этих случаях существуют тривиальные аналитические решения. В случае выбора функции MyFunct2 аналитическое решение существует в виде довольно замысловатой функции

Yinitial + ln(| tg(x / 2) / tg(Xinitial / 2) |)

а в случае функции f3 аналитического решения не существует, и единственным способом остаются непосредственные численные расчёты. Следующий рис.2.3.9 иллюстрирует решение дифференциального уравнения y'(x) = f3(x), где функция f3(x)=sin(x^2).

Созданное нами законченное приложение в рамках среды MATLAB удобно применять также для иллюстрации процесса сходимости приближённого решения к точному по мере увеличения числа шагов интегрирования N (то есть уменьшения шага интегрирования).

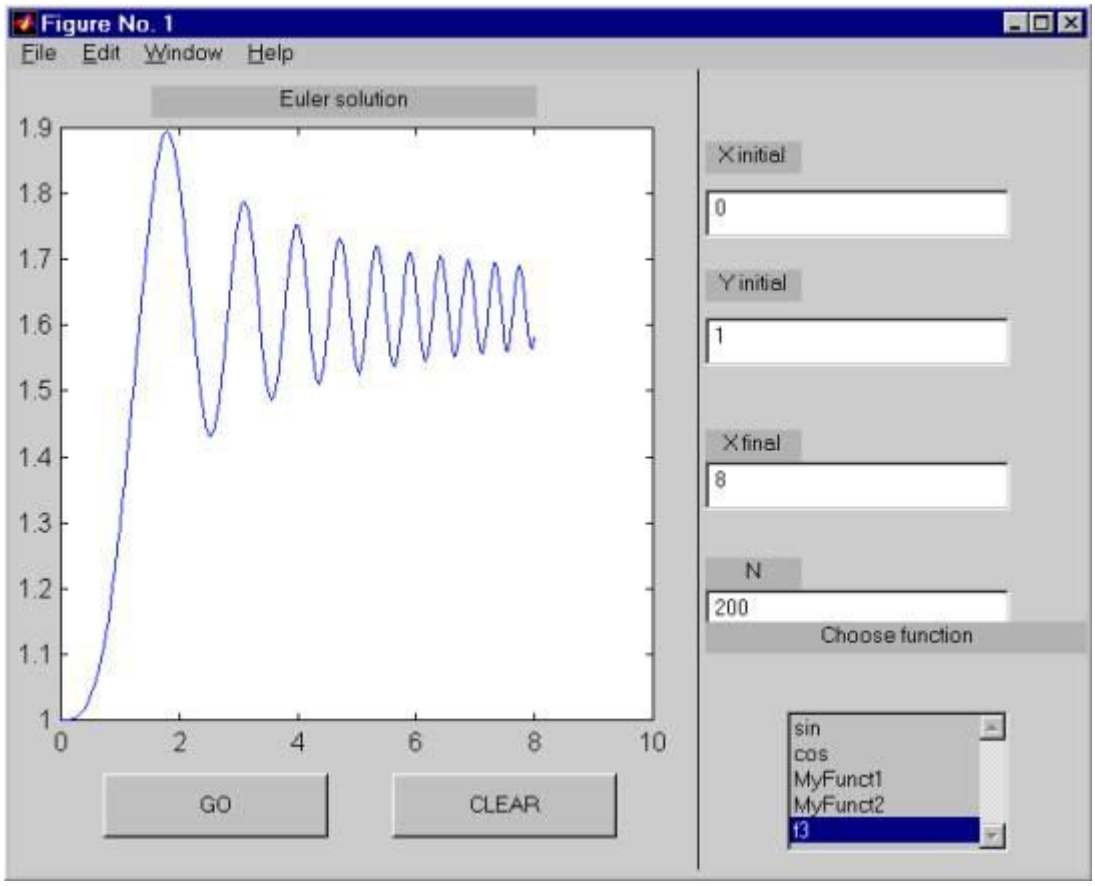


Рис.2.3.9. Графический интерфейс пользователя при вычислении функции f3, заданной пользователем

Заметим, что создание графических окон с элементами управления возможно не только с помощью "ручного" написания соответствующих М-файлов, изобилующих вызовами функций -конструкторов типа uicontrol, Axes и так далее. Этот способ, который мы рассмотрели подробно, является наилучшим с точки зрения максимального контроля разработчика над процессом разработки. В случае недостаточной квалификации разработчика (или необходимости очень быстрого выполнения работы) ему на помощь может прийти специальное визуальное средство разработки, входящее в пакет MATLAB - так называемый guide. По этой команде на экран дисплея выводится специальное окно, содержащее палитру графических элементов управления. С помощью мыши можно методом буксировки "перетаскивать" эти элементы на создаваемое собственное графическое окно. Правда callback-функции все равно нужно писать отдельно и самостоятельно. Изучение средства guide удобно проводить под руководством преподавателя на практическом занятии, повторяя демонстрируемые действия.

Часть III . Обработка изображений

3.1. 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 $f_r(x,y)$, $f_g(x,y)$, $f_b(x,y)$. По этой причине многие методы и приемы, разработанные для монохромных изображений, могут быть распространены на цветные изображения путем последовательной обработки трех монохромных компонент.

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 ProcessingToolbox (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.

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]

Чтобы ее увидеть, как изображение, используем команду >>imshow(A); %. Получим изображение слева на рис. 3.1.1.

Мы показали небольшую часть графического окна MATLAB.

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

>>B=mat2gray(A)

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

>>C=im2uint8(B)

>>imshow(C); % предыдущий рисунок справа

Рис. 3.1.1. Графическое представление матрицы *A* (слева) и преобразование значений пикселей изображения в диапазоне от 0 до 255 (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');

загружает файл 'football.jpg'.

Если в имя файла не включена информация о пути к данному файлу, то файл filename ищется в текущей папке.

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

size(f) % f содержит изображение football.jpg

Чтобы узнать формат графического файла можно использовать команду

info = imfinfo(' football.jpg’)

После обработки изображение 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 , 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');
```

3.2. Геометрические преобразования изображений

К наиболее распространенным функциям геометрических преобразований относится кадрирование изображений (imcrop), изменение размеров (imresize) и поворот изображения (imrotate). Суть кадрирования состоит в том, что функция imcrop позволяет с помощью мыши в интерактивном режиме вырезать часть изображения и поместить ее в новое окно просмотра. Например, запись:

```
L=imread('cameraman.tif');
imshow(L);
imcrop;
```

Применение операции кадрирования к изображению приведено на рис.3.2.1

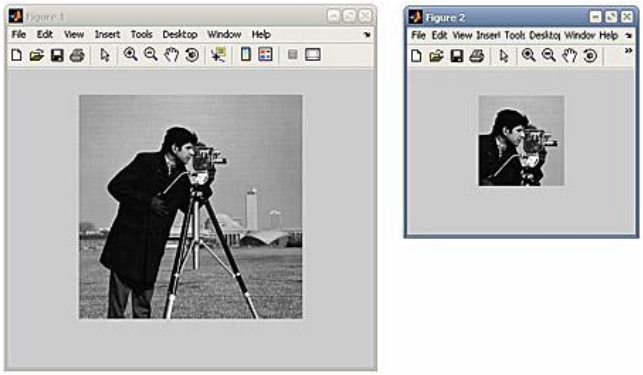


Рис. 3.2.1. Применение операции кадрирования к изображению cameraman.tif

Функция изменения размеров изображения imresize позволяет, используя специальные методы интерполяции, изменять размер любого типа изображения.

Изменение размеров изображения производится командой J=imresize(L,[mn]);

где L – исходное изображение,

[mn] – матрица количества пикселей по горизонтали и вертикали,

```
I = imread('cameraman.tif');
 J=imresize(I,[64 64]);
imshow(J)
```

Изменение размера изображения cameraman.tif показано на рис.3.2.2.

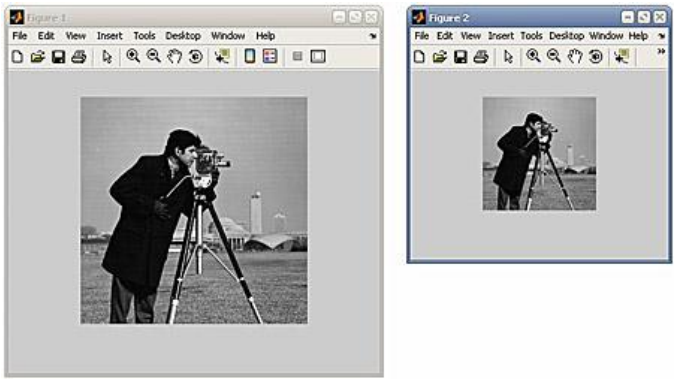


Рис. 3.2.2. Изменение размера изображения cameraman.tif

В пакете Image Processing Toolbox существует функция imrotate, которая осуществляет поворот изображения на заданный угол. Строки:

```
L1=imrotate(L,35,'bicubic');
figure, imshow(L1)
```

обеспечивают поворот изображения, что демонстрирует рис.3.2.3

Рис. 3.2.3. Поворот изображения cameraman.tif

Таким образом, приведенные выше функции позволяют поворачивать, вырезать части, масштабировать, т.е. работать с целым массивом изображения.

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

```
A = imread('Pogorelov32x32.bmp');
image(A)
C = double(A)/255;
C1=rgb2gray(C);
surf(C1)
```

Виды преобразований показаны на рис.3.2.4 – 3.2.6.

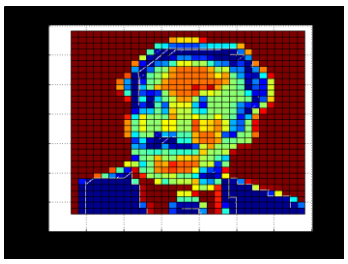


Рис. 3.2.4. Построение поверхности изображения Pogorelov32x32.bmp

surface(C1.*10); colormap(hot);

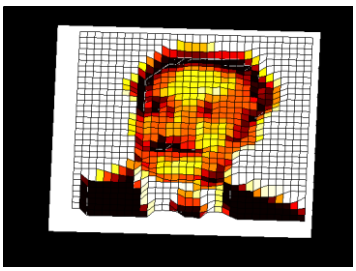


Рис.3.2.5. Изменение цвета палитры поверхности изображения Pogorelov32x32.bmp

surfc(C1);colormap(gray(32));

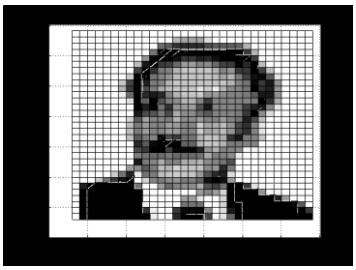


Рис. 3.2.6. Преобразование цвета поверхности изображения Pogorelov32x32.bmp в оттенки серого

Возможности изменения палитры матрицы демонстрирует следующий пример:

```
I = zeros(100,100);
I(25:75, 25:75) = 1;
imshow(I);
```

Виды преобразований показаны на рис.3.2.7 и рис. 3.2.8.

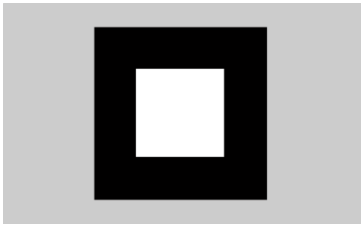


Рис. 3.2.7. Построение изображения матрицы

imshow(I);colormap winter

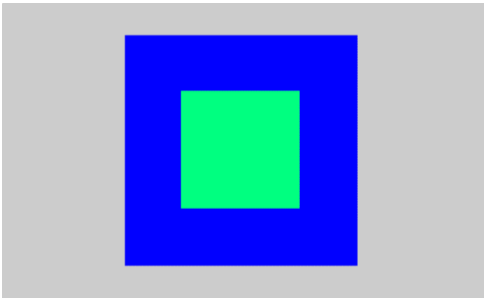


Рис. 3.2.8. Изменение цвета матрицы

Основа методов обработки изображений в Матлаб состоит в определении пространственной окрестности вокруг точки (*x*, *y*), которая может быть квадратной, прямоугольной или другой по форме областью с центром в точке (*x*, *y*), как показано на рис. 3.2.9.

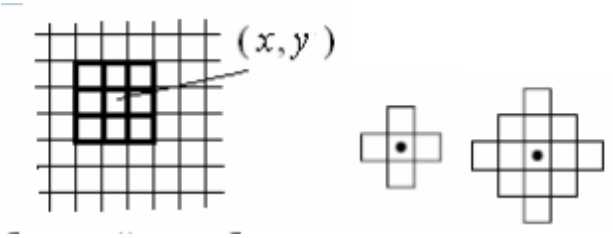


Рис. 3.2.9. Точка с координатами (*x*, *y*) и ее окрестности

Центр заданной шаблонной подобласти перемещается от пиксела к пикселу, начиная, из верхнего левого угла, и на своем пути накрывает различные окрестности. Преобразование применяется в каждой точке, 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 тогда, когда окрестность на рисунке имеет размер 1 x 1 (т.е. состоит из одного пиксела). В этом случае значение преобразованного изображения в точке (*x*, *y*) зависит только от значения *f* в этой точке, и *T* становится функцией преобразования яркости.

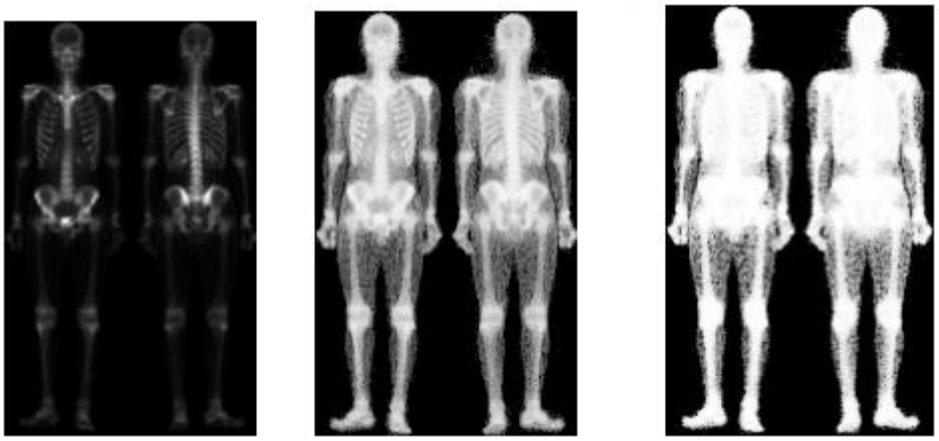
Пример 1. Усиление контрастности. На следующем рис. 3.2.10 слева показано входное изображение *f*. Применяя к нему преобразования

$$g_1 = \frac{f}{f + 10} \text{ и } g_2 = \frac{f^2}{f^2 + 5}$$

```
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)

мы получаем изображения, приведенные в центре и справа.



a) *b*) *c*)

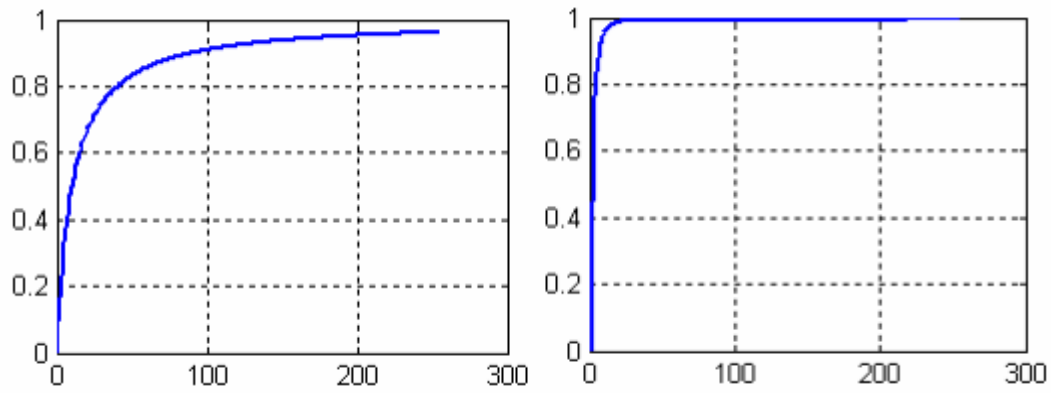
Рис. 3.2.10. Изображение Bone2.tif (*a*) и изменение его яркости по законам *g*1 (*b*) и *g*2 (*c*)

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

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;

имеют следующие графики – рис.3.2.11.



a) *b*)

Рис.3.2.11. Вид закона изменения яркости *g*1 (*a*) и *g*2 (*b*)

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))

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

$$g = c \cdot \log(1 + double(f))$$

где c – некоторая константа. Форма этой кривой при $c = 1$ и $0 \le f \le 1$ показана на рис. 3.2.12.

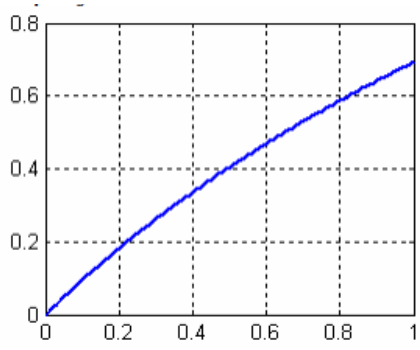


Рис.3.2.12. Закон изменения яркости вида *g=c*log(1+double(f))*

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

Пример 2. Использование логарифмического преобразования исходного изображения на рис.3.2.13.

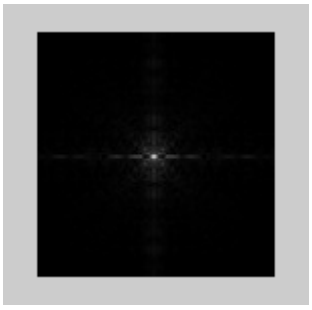


Рис. 3.2.13. Исходное изображение Spectrum2.tif

```
f=imread('Spectrum2.tif');
imshow(f);
size(f)
ans =
  122   118     3
f1=rgb2gray(f);
J=imresize(f1,[118 118]);
```

```
max(max(J))
ans =
  189
%Построим его значения по некоторым горизонтальным прямым
x=[1:1:118];
plot(x,J(59,:),x,J(40,:),x,J(30,:),x,J(20,:)); grid on;
```

График, показывающий изменения яркости, приведен на рис. 3.2.14.

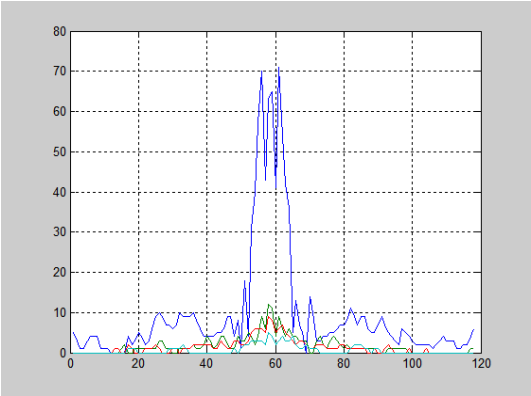


Рис. 3.2.14. Изменение яркости на изображении Spectrum2.tif

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. Далее на рис.3.2.15 показан результат применения команды логарифмического преобразования изображения

```
g = im2uint8(mat2gray(log(1+double(J))));
imshow(g)
```

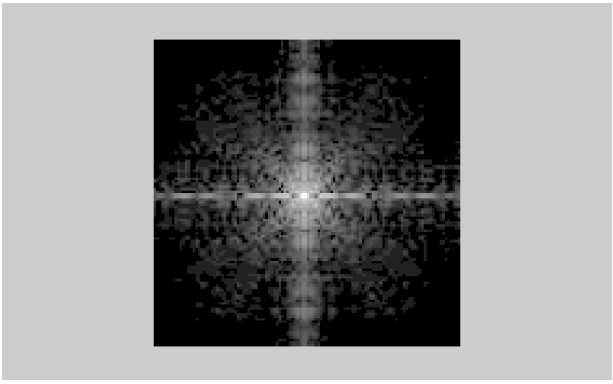


Рис. 3.2.15. Изображение Spectrum2.tif после логарифмического преобразования яркости

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

файл 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
% Функция преобразования
z=@(x) x/20.*(x<20)+ ((x-180).^4/160^4).*(x>=20 & x<=180) +(x-180)/175.*(x>180);%
писать выражение в одну строку
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
```

ex71.m

```
%программа адаптирована к матлаб 6.5
% Функция преобразования
x=0:1:255;
z=x/20.*(x<20)+ ((x-180).^4/160^4).*(x>=20 &x<=180) +(x-180)/175.*(x>180);
plot(x,z,'LineWidth',2); grid on; set(gcf,'Color',[1 1 1]);
axis([0 255 0 1.1]);
% исходное изображение
f=imread('Phobos2.jpg'); % чтение исходного изображение
figure,
imshow(f);
```

```
x=double(f);
z=x/20.*(x<20)+ ((x-180).^4/160^4).*(x>=20 & x<=180) +(x-180)/175.*(x>180);
g1=z.*255;
figure;g=mat2gray(g1);  imshow(g);  % результирующее изображение
```

Результаты преобразований показаны на рис. 3.2.16 и рис.3.2.17.

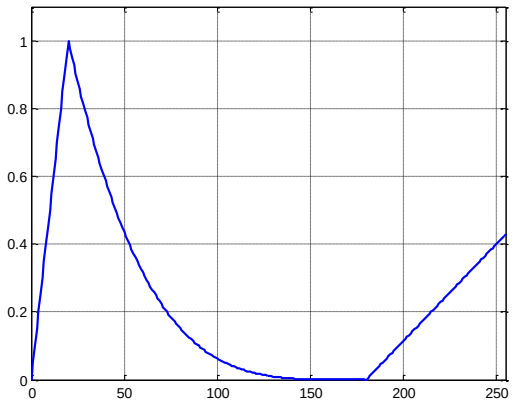


Рис 3.2.16. Закон преобразования яркости для изображения Phobos2.jpg

a) *b*)

Рис. 3.2.17. Исходное изображение Phobos2.jpg (*a*) и преобразованное изображение (*b*)

3.3. Анализ изображений

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

Гистограммой цифрового изображения называется дискретная функция

$h(rk)=nk$,

где rk – это k– й уровень яркости (из интервала квантованных яркостей[0,G]), а nk – число пикселов изображения, уровень яркости которых равен rk. Значение G равно255 для изображений класса uint8, 65535 – для классauintl6 и1.0 – для класса double.

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

h = imhist(f, b),

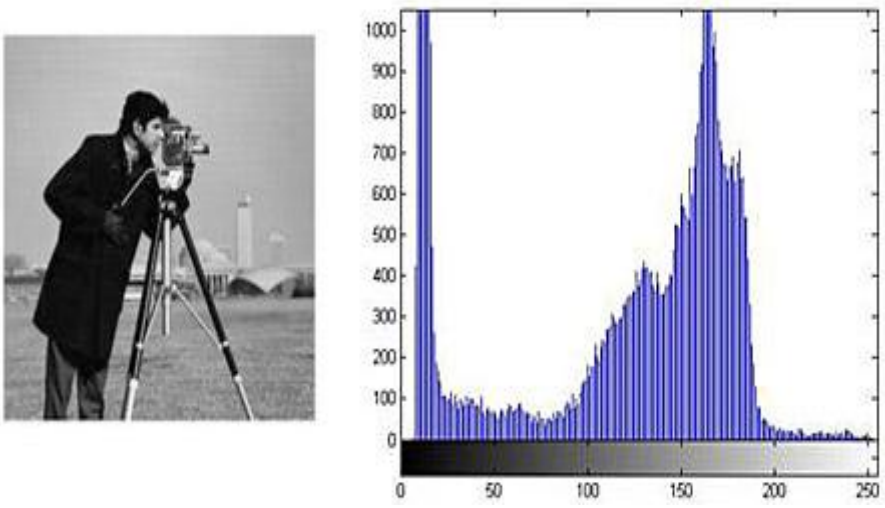
где *f* – это входное изображение, *h* – его гистограмма *h*(*rk*) (фактически вектор значений), и *b* – число корзин, использованных при формировании гистограммы (если аргумент *b* отсутствует, то по умолчанию принимается *b*= 256). Корзиной называется подразделение шкалы яркости. Например, если при работе с изображениями класса uint8 переменная *b* = 2, то шкала яркости делится на две подобласти (корзины): от 0 до127 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.

Например,преобоазование

```
L=imread('cameraman.tif');
figure, imshow(L);
figure, imhist(L);
```

показано на рис.3.3.1.



a) *b*)

Рис. 3.3.1. Изображение cameraman.tif *(a*) и его гистограмма (*b*)

Довольно часто при проведении анализа изображений возникает необходимость определить значения интенсивностей некоторых пикселей. Для этого в интерактивном режиме можно использовать функцию impixel.

```
impixel
ans =
  173   173   173
```

169 169 169
163 163 163
39 39 39

3.4. Улучшение изображений

Среди встроенных функций, которые реализуют наиболее известные методы улучшения изображений, выделим следующие – histeq, imadjust та imfilter(fspecial).
Гистограмма изображения является одной из наиболее информативных характеристик. На основе анализа гистограммы можно судить о яркостных искажениях изображения, т.е. сказать о том, является ли изображение затемненным или за светленным. Известно, что в идеале на цифровом изображении в равном количестве должны присутствовать пиксели со всеми значениями яркостей, т.е. гистограмма должна быть равномерной. Перераспределение яркостей пикселей на изображении с целью получения равномерной гистограммы выполняет метод эквализации, который в системе Matlab реализован в виде функции histeq. Фрагмент программы преобразует исходный файл (рис.3.4.1) в вид, представленный на рис.3.4.2.

```
L=imread('cameraman.tif');
figure, imshow(L);
L1=histeq(L);
figure, imshow(L1);
```

Рис. 3.4.1. Исходное изображение

Рис. 3.4.2. Изображение после эквализации гистограммы

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)

g=rgb2gray(f); f1=histeq(g, 256); imshow(f1); % эквализация

Напишем файл-сценарий **ex8.m**, который модифицирует гистограмму исходного изображения и позволяет рассмотреть более четко детали изображения

```
% исходное изображение
f1=imread('Phobos2.jpg');% чтение исходного изображения
f=rgb2gray(f1);
h0=imhist(f); x = 1:1:256;
bar(x, h0, 0); gridon;  % 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,
чтобыsum(h1)=numel(f)
figure,bar(x, h1, 0); grid on;
34
axis([0 255 0 5000]); set(gcf,'Color',[1 1 1]);
figure,g = histeq(f, h1); % генерируем изображение с желаемой
гистограммойh1
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;
```

ex81.m

```
% программа адаптированная для матлаб 6.5
% исходное изображение
f1=imread('Phobos2.jpg');% чтение исходного изображения
f=rgb2gray(f1);
h0=imhist(f); x = 1:1:256;
bar(x, h0, 0); gridon;  % гистограмма исходного изображения
axis([0 255 0 20000]); set(gcf,'Color',[1 1 1]);
% Построение ломаной, моделирующей форму желаемой гистограммы
x=0:0.1:255;
z=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));
figure,plot(x,z,'LineWidth',2); grid on; set(gcf,'Color',[1 1 1]);
axis([0 255 0 75000]);
% моделируем желаемую гистограмму
h01=z;
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 гистограммойh1
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;
```

Довольно часто при формировании изображений не используется весь диапазон значений интенсивностей, что отрицательно отражается на качестве визуальных данных. Для коррекции динамического диапазона сформированных изображений используется функция imadjust.

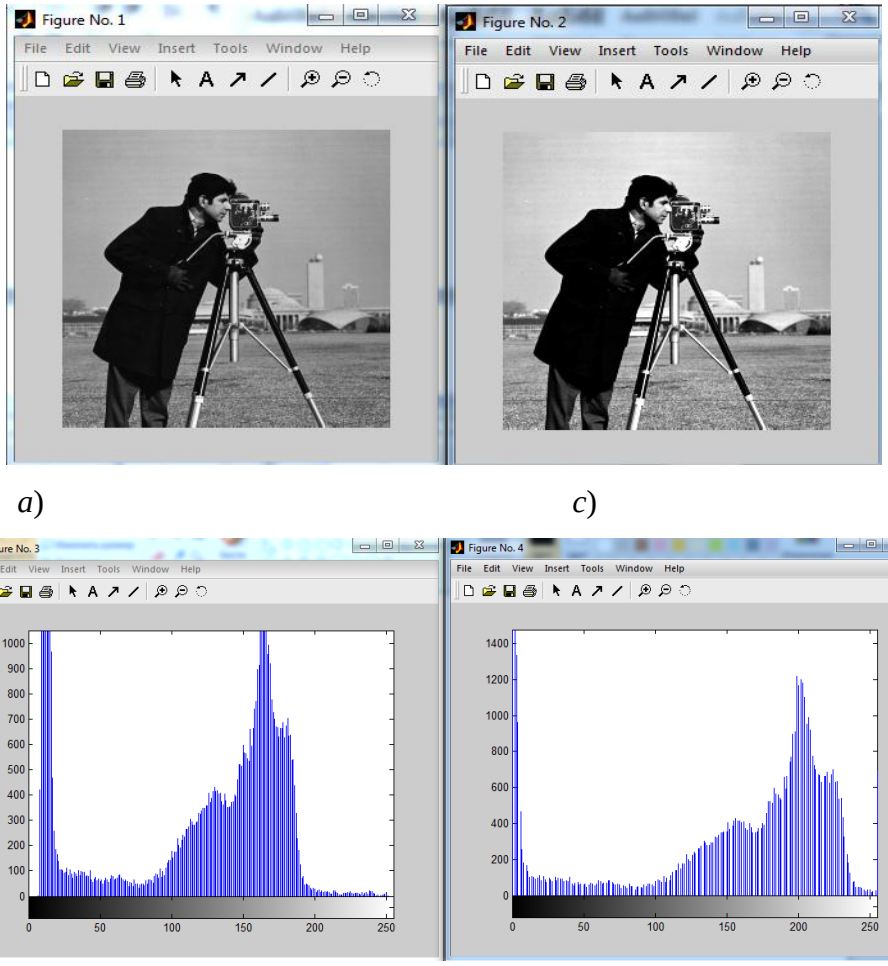
Функция imadjust является базовым инструментом пакета IPT при преобразованиях яркости полутоновых изображений. Она имеет следующий синтаксис: g=imadjust(f,[low_in,high_in],[low_out,high_out],gamma) Она выполняет преобразование яркости = (fTg) исходного изображения f в новое изображение g, при котором 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, uintl6 или double, и класс выходного изображения совпадает с классом входного. Это значит, что элементы матрицы изображения (монохромного) являются без знаковыми целыми числами, изменяющимися в диапазоне 0 – 255 (uint8); без знаковыми целыми числами, изменяющимися в диапазоне 0 – 65535 (uint16); или вещественными числами (double) обычно изменяющимися в диапазоне [0 – 1].

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.

Фрагмент преобразования:

```
L=imread('cameraman.tif');
figure, imshow(L);
L1=imadjust(L, [0.05 0.8],[]);
figure, imshow(L1);
figure, imhist(L);
figure, imhist(L1);
```

приводит к видам, изображений, приведенных на рис. 3.4.3.



a) *c*)

b) *d*)

Рис..3.4.3. Исходное изображение cameraman.tif (*a*), его гистограмма (*b*) и преобразованное изображение с помощью функции imadjust (*c*) с гистограммой (*d*)

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.

3.5. Сегментация изображений

Среди встроенных функций пакета ImageProcessingToolbox, которые применяются при решении задач сегментации изображений, следует выделить qtdecomp, edge и roicolor. Функция qtdecomp выполняет сегментацию изображения методом разделения и анализа однородности не перекрывающихся блоков изображения.

Функция **qtdecomp** осуществляет сегментацию полутоновых изображений методом разделения. Суть метода заключается в следующем. Изображение разбивается на неперекрывающиеся блоки. Каждый блок с помощью некоторого критерия проверяется на однородность. Если блок неоднороден, то он разбивается на блоки меньшего размера, каждый из которых, в свою очередь, проверяется на однородность. Процесс завершается тогда, когда ни один из блоков не может быть разделен, т. е. либо блоки однородны, либо их размеры достигли предельно малых величин. В результате работы алгоритма получается набор однородных блоков различного размера.

В функции **qtdecomp** каждый блок разбивается на 4 не перекрывающихся блока одинакового размера. На первом шаге алгоритма блоком считается все изображение. Мельчайшим по размерам является блок, который нельзя разделить на 4 блока одинакового размера, т. е. такой блок, у которого число строк или число столбцов нечетное. Таким образом, в функции **qtdecomp** рекомендуется использовать изображения с размерами, равными степеням двух. В этом случае мельчайший блок будет состоять из одного пикселя. При реализации алгоритма разделения используются структуры данных, основанные на квадро-деревьях. По этой причине данный алгоритм часто называют декомпозицией или разделением с помощью квадро-деревьев.

Функция **A=qtdecomp(I)** осуществляет сегментацию полутонового изображения *I* методом разделения и помещает результат в разреженный массив *A* (тип данных sparse MATLAB). Разреженный массив *A* конструируется следующим образом. Элементам матрицы $A(r, c)$, соответствующим координатам левых верхних углов блоков на исходном изображении *I*, присваиваются значения, определяющие размеры каждого блока. Таким образом, большинство элементов матрицы равно нулю. И поэтому для хранения квадро-дерева применяется разреженный массив, который эффективно использует память, когда большинство элементов массива равно нулю. Для данной функции критерием однородности блока является равенство всех пикселей блока друг другу.

Функция **A=qtdecomp(I, threshold)** работает аналогично описанной выше, но в ней блок считается однородным, если разница между максимальным и минимальным значением пикселей блока меньше параметра threshold.

В функции **A=qtdecomp(I, threshold, mindim)** параметр mindim определяет минимальный размер блока. Неоднородный блок не разделяется, если в результате разделения получаются блоки размером менее mindim.

В функции **A=qtdecomp(I, threshold, [mindimmaxdim])** параметр mindim определяет минимальный, а параметр maxdim - максимальный размер блока. Неоднородный блок не разделяется, если в результате разделения получаются блоки размером менее mindim. Блок, превышающий размером maxdim, разделяется, даже если он однородный. Отношение параметров maxdim/mindim должно быть равно степени двух.

Функция **A=qtdecomp(I, fun)** позволяет для определения однородности блока передать вторым параметром собственную функцию fun. Существует 3 варианта задания параметра fun. Они приведены в таблице в описании функции **blkproc**. В функцию fun передается массив B из всех текущих блоков размера $m \times m$, т. е. массив $m \times m \times k$, где k – количество блоков. В результате работы функции **fun** возвращается вектор Z длиной k элементов, состоящий из 0 и 1. Единица указывает на то, что соответствующий блок должен быть разделен, а нуль - на то, что блок не следует далее разделять, т. е. если $Z(i) = 1$, то блок $B(:,:, i)$ должен быть разделен.

Функция **A=qtdecomp(I, fun, PI, P2, ...)** позволяет передать в функцию fun дополнительные параметры PI, P2,...

Пример:

```
I = imread('cameraman.tif');
S = qtdecomp(I,.27);
imshow(S)
```

Сегментированное изображение cameraman.tif показано на рис.3.5.1.

Рис. 3.5.1. Сегментированное изображение cameraman.tif

Пример с визуализацией блоков разделения изображения реализуется программой

```
I = imread('cameraman.tif');
S = qtdecomp(I,.27);
blocks = repmat(uint8(0),size(S));
```

```
for dim = [512 256 128 64 32 16 8 4 2 1];
numblocks = length(find(S==dim));
if (numblocks> 0)
values = repmat(uint8(1),[dim dim numblocks]);
values(2:dim,2:dim,:) = 0;
blocks = qtsetblk(blocks,S,dim,values);
end
end
blocks(end,1:end) = 1;
blocks(1:end,end) = 1;
imshow(I), figure, imshow(blocks,[])
```

и рис. 3.5.2.

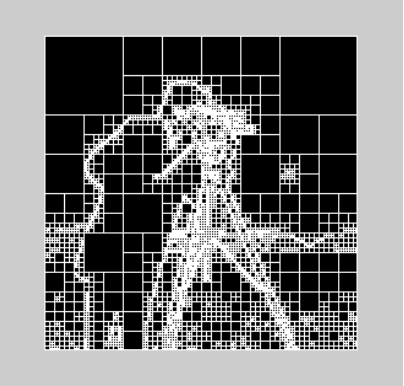


Рис. 3.5.2. Сегментированное изображение cameraman.tif с визуализацией блоков

Одной из наиболее часто применяемых является функция выделения границ edge, которая реализует такие встроенные методы – Собела, Превит, Робертса, лапласиан-гауссиана, Канни и др.

Примеры реализации функции edge с использованием различных фильтров:

```
clear;
I = imread('cameraman.tif');
BW1=edge(I,'sobel');
figure,imshow(BW1);title('sobel');
BW2=edge(I,'prewitt');
figure,imshow(BW2);title('prewitt');
BW3=edge(I,'roberts');
figure,imshow(BW3);title('roberts');
BW4=edge(I,'log');
figure,imshow(BW4);title('log');
```

BW5=edge(I,'zerocross');

figure,imshow(BW5);title('zerocross');

BW6=edge(I,'canny');

представлены на рис.3.5.3.

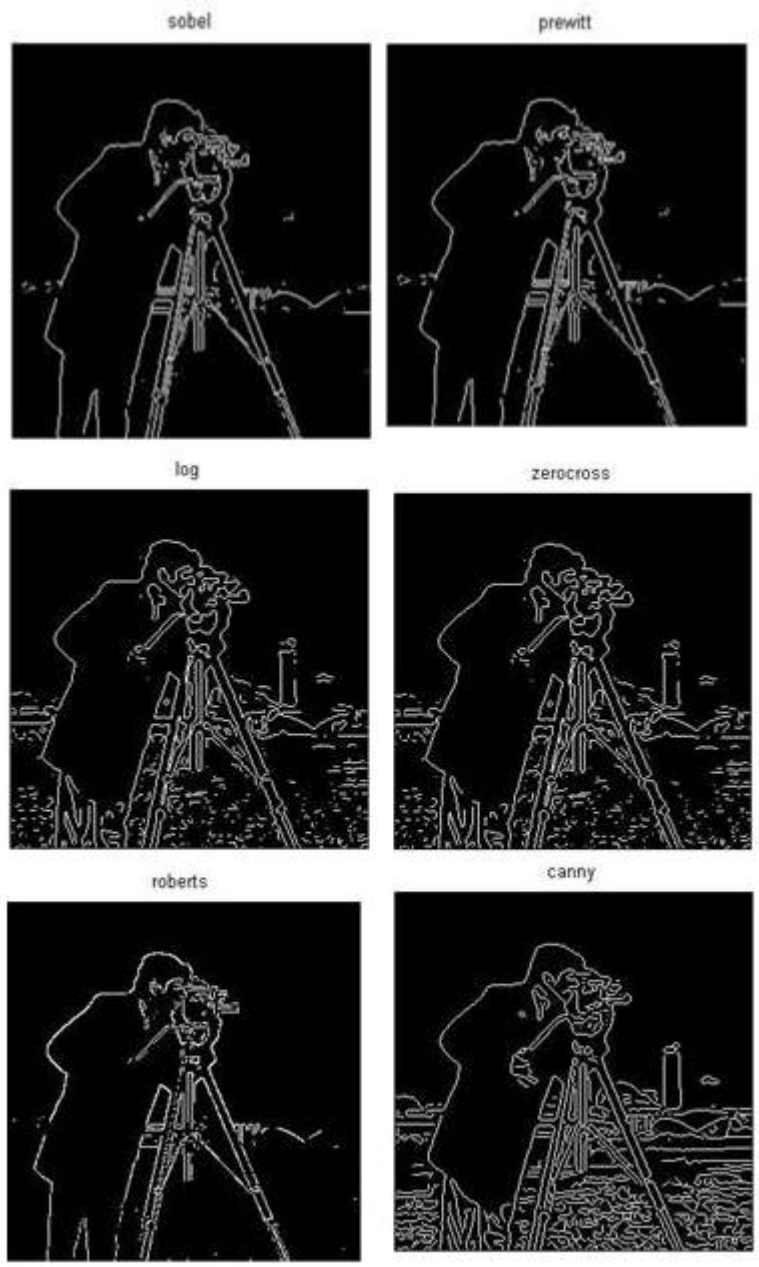


Рис.3.5.3. Фильтрация изображения cameraman.tif с помощью различных фильтров

Еще одной функцией, которая применяется для бинаризации по заданным цветам, является функция roicolor. Приведем пример ее использования (рис.3.5.4).

I = imread('cameraman.tif');

```
figure, imshow(I);
BW = roicolor(I,128,255);
imshow(I);
figure, imshow(BW);
```

a) *b*)

Рис. 3.5 4. Исходное (*a*) и бинаризованное (*b*) изображение

3.6. Морфологические операции над бинарными изображениями

Система Matlab владеет довольно мощным инструментарием морфологической обработки бинарных изображений. Среди основных операций выделим следующие – эрозия, наращивание, открытие, закрытие, удаление изолированных пикселей, построение скелета изображения и т.п.

Функция **BWd=bwmorph(BWd, operation)** создает бинарное изображение BWd, подвергая обработке морфологическим фильтром исходное бинарное изображение BWs. Тип используемого морфологического фильтра определяется параметром operation. Морфологические операции (их также называют морфологическими фильтрами) представляют собой нелинейный способ обработки изображений. Их применяют с целью изменения формы объектов. Рассматривая морфологические операции, будем называть связные области, значения пикселей которых равны единице, объектами, а области изображения, значения пикселей которых равны нулю, фоном.

Параметры operation и их назначение приведены в табл. 3.6.1.

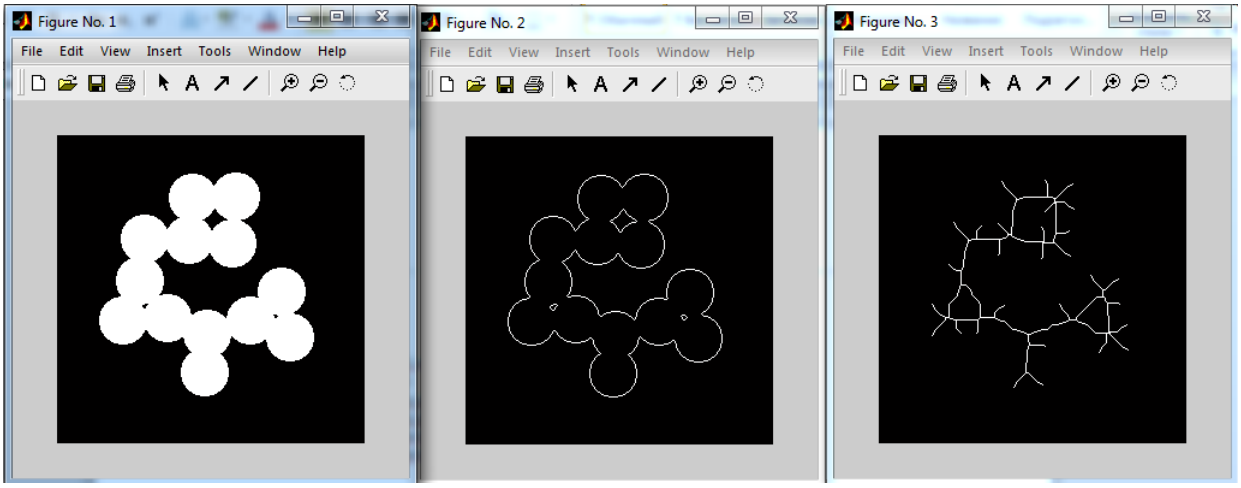
Таблица 3.6.1. Опции морфологических фильтров

Параметр	Назначение параметра
"erode"	*Эрозия объекта.* Приводит к замене значений граничных пикселей объекта на 0. Однократное применение эрозии приводит к удалению слоя границы толщиной в 1 пиксель (рис. 3.6.1,*b).*
"dilate"	*Наращение объекта.* Приводит к замене значений пикселей фона, граничащих с объектом, на 1. Однократное применение наращения приводит к добавлению к объекту слоя толщиной в 1 пиксель (рис. 3.6.1, *c).*
"open"	*Открытие.* Представляет собой последовательное применение эрозии и наращения. Приводит к соединению областей фона, ранее разъединенных узкими участками пикселей объектов
"close"	*Закрытие.* Представляет собой последовательное применение наращения и эрозии. Приводит к удалению небольших по площади фрагментов фона внутри объектов, например "дыр" (замкнутых областей фона внутри объекта) .
"tophat"	*Преобразование типа "верх шляпы ".* Соответствует вычитанию из исходного изображения результата его открытия
"bothat"	*Преобразование типа "низ шляпы".* Соответствует вычитанию исходного изображения из результата его закрытия
"clean"	*Удаление изолированных пикселей объектов.* Пиксели равные 1, все 8 соседей, которые не равны 0, заменяются на 0.
"fill"	*Заполнение изолированных пикселей фона.* Пиксели, равные 0, все 8 соседей которые не равны 1, заменяются на 1.
"diag"	*Уничтожение 8-связности фона.* Осуществляется добавлением необходимого количества единиц во фрагменты объектов, связанных только по диагонали. Например, матрица фрагмента изображения $\begin{bmatrix}0&1&0\\1&0&0\\1&0&0\end{bmatrix}$ заменяется на $\begin{bmatrix}0&1&0\\1&1&0\\1&0&0\end{bmatrix}$
"bridge"	*Соединение пикселей объекта*, разъединенных фрагментом фона толщиной в 1 пиксель. Например, матрица фрагмента изображения $\begin{bmatrix}0&0&1\\1&0&1\\1&0&0\end{bmatrix}$ заменяется на $\begin{bmatrix}0&0&1\\1&1&1\\1&0&0\end{bmatrix}$
"hbreak"	Удаление центрального пикселя в конфигурациях, похожих на букву "H": $\begin{bmatrix}1&0&1\\1&1&1\\1&0&1\end{bmatrix}$ и $\begin{bmatrix}1&1&1\\0&1&0\\1&1&1\end{bmatrix}$ заменятся соответственно на $\begin{bmatrix}1&0&1\\1&0&1\\1&0&1\end{bmatrix}$ и $\begin{bmatrix}1&1&1\\0&0&0\\1&1&1\end{bmatrix}$

Продолжение табл.3.6.1.	
"remove"	*Удаление внутренних пикселей объектов.* Для этого в 0 устанавливаются пиксели объекта, у которых 4 соседних по горизонтали и вертикали пикселя были равны 1, т.е. тоже являлись пикселями объекта. В результате применения этого морфологического фильтра не сброшенными останутся только пиксели границы объекта (рис. 18.1, *b*).
"majority"	*Эрозия и наращение по преобладанию* в окрестности пикселей фона или объекта. Осуществляется следующим образом: если в окрестности пикселя размера 3x3 находится 5 или более пикселей объекта, то рассматриваемый пиксель устанавливается в 1, в противном случае – в 0.
"skel"	*Построение остова (скелета) объекта.* Операция выполняет эрозию объекта с учетом ряда условий для сохранения 8-связности остова. В результате последовательного применения данной операции можно построить остов, представляющий собой связную линию толщиной в 1 пиксель, проходящую по середине объекта (рис. 18.1, *c*).
"shrink"	*Сжатие объекта.* Операция выполняет эрозию объекта с учетом ряда условий для сохранения 8-связности замкнутых участков остова. В результате последовательного применения данной операции объекты, не содержащие дыр, превращаются в точку, а объекты с дырами "сжимаются" в 8-связные замкнутые участки остова, проходящие посередине и по внешней границе объекта.
"thin"	*Утоньшение объекта.* Операция выполняет эрозию объекта с учетом ряда условий для сохранения 8-связности участков остова. В результате последовательного применения данной операции объекты, не содержащие дыр, превращаются в одну или несколько связных линий с минимальным количеством разветвлении остова по сравнению с остовом, получаемым с помощью оператора 'skel", а объекты с дырами сжимаются в 8-связные замкнутые участки остова, проходящие посередине между границами дыр и внешней границей объекта .
"thicken"	*Утолщение объекта.* Операция выполняет наращение объекта с учетом ряда условий для сохранения 4-связности участков фона. Данная операция может рассматриваться как построение остова фона и является по смыслу обратной операции 'thin'. Результат данной операции будет незначительно отличаться от результатов применения оператора 'thin' к инвертированному изображению.
"spur"	*Удаление ответвлений объекта* толщиной в 1 пиксель, т. е. удаление пикселей, у которых только один из соседних пикселей установлен в 1, а остальные - в 0.Например, фрагмент изображения $\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}$ заменяется на $\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}$.В результате последовательного применения данной операции к изображению остова на изображении остаются только замкнутые участки остова.

Пример:

```
BW1 = imread('circles.tif');
imshow(BW1)
      BW2 = bwmorph(BW1,'remove');
      BW3 = bwmorph(BW1,'skel',Inf);
figure, imshow(BW2)
figure, imshow(BW3)
```



a) *b*) *c*)

Рис.3.6.1.Исходное изображение *(a*), изображение с удаленными внутренними пикселами (*b*), остов («скелет») объекта (*c*)

3.7. Стеганография в цифровых изображениях

Стеганография в цифровых изображениях – раздел графики, изучающий проблему сокрытия данных в цифровых изображениях. В отличие от криптографии, задача стеганографии –скрыть сам факт наличия скрытого сообщения. Основные методы сокрытия информации в цифровых изображениях можно разделить на: пространственные и частотные.

Задача стеганографии в изображениях – встроить информацию в цифровое изображение так, чтобы и сообщение, и сам факт его наличия был скрыт. Полученное изображение с дополнительной скрытой информацией не должно выглядеть аномальным. Это достигается путём внесения изменений, незаметных для человеческого зрения. Многие методы стеганографии используют методики, схожие с методами сжатия изображений.

На практике методы стеганографии применяются для идентификации, защиты авторских прав и сокрытия передаваемых сообщений.

Хранение изображений в цифровом формате упрощает их хранение и распространение, но так же увеличивает риск нарушения авторских прав, не санкционированного изменения и распространения. В целях защиты интеллектуальной собственности и определения изменений, разрабатываются и применяются цифровые водяные знаки. К таким стеганографическим методикам предъявляются особые требования:

Качество исходного изображения не должно быть серьёзно затронуто, скрытые данные должны быть минимально заметны.

Скрытые данные должны сохраняться в разных форматах, то есть содержаться не только в заголовке, а во всем теле изображения.

Скрытые данные должны быть устойчивы к намеренным попыткам удаления.

Необходимо наличие избыточного кода для коррекции ошибок, так как деградация данных при передаче/модификации неизбежна.

В 2018 году сотрудник General Electric использовал фотографию заката чтобы украсть 40 файлов Excel и Matlab, содержащих данные, являющиеся коммерческой тайной.

Facebook манипулирует метаданными изображений, публикуемых на сайте, для отслеживания их дальнейшего распространения.

Цифровые стего-изображения так же встречаются в известной интернет-головоломке Цикада 3301, одним из основных фокусов которой является стегоанализ.

Пространственные методы стеганографии в цифровых изображениях манипулируют значениями в пространственной области (пикселями).

Метод LSB (англ. Least Significant Bit — Наименеезначимый бит)

Данный метод заключается в выделении наименее значимых бит изображения-контейнера с последующей их заменой на биты сообщения. Поскольку замене подвергаются лишь наименее значимые биты, разница между исходным изображением-контейнером и контейнером, содержащим скрытые данные невелика и обычно незаметна для человеческого глаза. Метод LSB применим лишь к изображениям в форматах без сжатия (например, BMP) или со сжатием без потерь (например, GIF), так как для хранения скрытого сообщения используются наименее значимые биты значений пикселей, при сжатии с потерями эта информация может быть утеряна. Форматы без сжатия имеют очень большой размер и могут вызвать подозрение, по этому для стеганографии чаще используют другие форматы.

Метод Встраивание

Например, имеется чёрно-белое изображение, представленное в виде матрицы M[i,j]. Значения в этой матрице соответствуют яркостям пикселей, расположенных по координатам (i,j). Пусть эти значения представлены восьмибитными двоичными числами. Пусть скрываемое сообщение имеет размер 2 байта. Для хранения скрытого сообщения возьмём 2 младших бита изображения-контейнера. Тогда для сокрытия 2 байт, то есть 16 бит необходимо изображение размером минимум 8 пикселей. Если изображение содержит больше пикселей, чем необходимо для хранения сообщения, необходимо выбрать правило, по которому будут выбираться пиксели для встраивания данных. Этот закон должен быть заранее известен получателю, так как он будет необходим для извлечения данных. Для сокрытия факта встраивания данных, к неиспользованным пикселям изображения добавляется шум, чтобы шум, вносимый скрытыми данными, не выглядел аномальным. Например, имея изображение размером 4x2 пикселя запишем первые два бита сообщения b=0001101100011011 в первый пиксель: пусть исходное значение пикселя M[i,j]= 10100101, заменим младшие два бита на первые два бита сообщения M[i,j]= 10100100. Следующие два бита записываются в следующий пиксель, и так далее.

Рассмотрим пример встраивания сообщения в изображение-контейнер с использованием Матлаб.

Для кодирования необходимо выполнить следующие операции:

1. Преобразовать изображение в оттенки серого.

2. При необходимости изменить размер изображения.

3. Преобразовать сообщение в его двоичный формат.

4. Инициализировать выходное изображение так же, как входное.

5. Просмотреть каждый пиксель изображения и сделать следующее:

• Преобразовать значение пикселя в двоичное.

• Получить следующий бит сообщения для встраивания

• Создатьвременную переменную temp

• Если бит сообщения и младший бит пикселя совпадают, установить temp = 0

• Если бит сообщения и младший бит пикселя отличаются, установить temp = 1

• Эту настройку temp можно выполнить, взяв XOR бита сообщения и младшего разряда пикселя.

• Обновить пиксель выходного изображения до значения пикселя входного изображения +temp.

6. Продолжать обновлять выходное изображение, пока все биты в сообщении не будут встроены.

7. Наконец, записать входное и выходное изображение в локальную систему.

Листинг программы Матлаб

```
% Очистка рабочей области
clear all;

% Очистка окна команд
clc;

% Считывание изображения-контейнера
input = imread('Penguins.jpg');

%  Преобразование входного изображения в оттенки серого
input=rgb2gray(input);

% Изменение размеров изображения
input=imresize(input, [512 512]);

% встраиваемоесообщение
message='Dancing Penguins';

% Вычисление длины сообщения, где каждый символ занимает 8 бит
len = length(message) * 8;

% Получение значений  ASCII – кодов символов сообщения
ascii_value = uint8(message);

% Преобразование десятичных значений в двоичные
bin_message = transpose(dec2bin(ascii_value, 8));

%  вывод двоичных цифр в отдельную строку
bin_message = bin_message(:);

%   вычисление длины бинарного сообщения
N = length(bin_message);

% Преобразование символного массива в числовой
bin_num_message=str2num(bin_message);

% Обозначение выходного изображения как входного
output = input;

% Получить высоту и ширину для перемещения по изображению
height = size(input, 1);
width = size(input, 2);

% Счетчик количества встроенных битов
embed_counter = 1;

% Проход по изображению
```

```
for i = 1 : height
for j = 1 : width

% Проверка, остались ли еще биты для вставки
if(embed_counter <= len)

% Нахождение наименее значимого бита (Least Significant Bit)
текущего пикселя
LSB = mod(double(input(i, j)), 2);

% Определение, является ли бит тем же самым, или его нужно
поменять
temp = double(xor(LSB, bin_num_message(embed_counter)));

% Обновление выходного изображения до вход+  temp
output(i, j) = input(i, j)+temp;

% Увеличение счетчика встраиваемых битов
            embed_counter = embed_counter+1;
end

end
end

% Запись входного и выходного изображений в локальное хранилище
% Нужно указать путь к папке.
imwrite(input, 'D:originalImage.png');
imwrite(output, 'D:stegoImage.png');
```

Реализация программы приведена на рис.3.6.2.

a) *b*)

Рис.3.6.2. Исходное изображение-контейнер (*a*) и изображение со встроенным сообщением (*b*)

Метод Извлечение

Для извлечения скрытых методом LSB данных, необходимо выбрать пиксели, содержащие полезную нагрузку по тому же закону, по которому они выбирались при встраивании. Далее, имея набор пар координат вида (i,j), по очереди, извлекаются наименее значимые биты: $M[i,j]$= 10100100. Извлечённые биты данных объединяются, формируя скрытое сообщение.

PVD (англ. Pixel Value Difference – Разность значений пикселей

Этот метод учитывает тот факт, что на гладких участках (где значение яркости меняется незначительно) изменение будет более заметно, нежели на участках, содержащих более значительные перепады яркости.

Встраивание

Исходное изображение разделяется на блоки по 2 пикселя, и скрытые данные кодируются как разность значений внутри этих блоков. Как и в случае с LSB, необходим закон, по которому будут выбираться блоки для встраивания. Для каждого используемого блока вычисляется модуль разности значений пикселей, по которому определяется диапазон допустимых значений. Чем больше перепад яркости внутри блока- тем шире выбранный диапазон. Для удобства работы, ширина диапазона является степенью двух. Тогда, например, в блок с диапазоном шириной $4=2^2$ можно записать 2 бита скрываемого сообщения (эти два бита, по сути, представляют собой выбор конкретного числа из диапазона). Блоки, изменение которых может привести к выходу за пределы допустимых значений яркости пикселей (от 0 до 255) не используются.

Извлечение

Для извлечения данных, изображение вновь делится на блоки по 2 пикселя. В соответствии с заранее известным правилом выбора блоков и последовательностью их обхода, для блоков рассчитывается разность значений пикселей и определяется диапазон, в который она попадает. Далее выполняется проверка на выход за пределы диапазона от 0 до 255: если при максимальной разности, входящий в диапазон, один из пикселей принимает значение больше 255 или меньше 0, то данный блок пропускается, так как он был отброшен аналогичной проверкой на стадии встраивания. Из оставшихся блоков извлекаются данные: по ширине диапазона определяется количество бит, встроенных в блок, которые потом извлекаются, начиная с наименее значимого. В общем же случае, когда ширина диапазона не является степенью 2, полезная нагрузка вычисляется как $b=|g_i-g_{i+1}|-l_k$,где b – извлекаемые данные, g_i,g_{i+1}– значения пикселей блока, l_k- нижняя граница диапазона.

GLM (англ. Grey Level Modification — Изменение уровня серого)

Метод GLM заключается в изменении чётности значения яркости изображения в чёрно-белом представлении. В каждый пиксель изображения встраивается 1 бит скрываемого сообщения.

Встраивание

В начале значения яркости всех пикселей делаются чётными, путём изменения всех нечётных значений на 1. Далее чётность этих значений сравнивается с чётностью битов данных. Например, если первый бит данных чётный (то есть равен 0), то первый пиксель не изменяется, если же он нечётный (равен 1), то значение яркости изменяется на нечётное.

Извлечение

Для каждого пикселя, содержащего скрытое сообщение, определяется значение яркости. Если оно чётное- то соответствующий бит сообщения равен 0, если нечётное — то 1.

MPV (англ. Mid Position Value — Значение в средней позиции)

В данном методе к изображению-контейнеру сначала применяется преобразование Арнольда, затем для каждого пикселя вычисляется его позиция $p(i,j)=(i\text{-}1)m+j$. Для каждого $p(i,j)$ оценивается количество знаков total_digits и положение среднего знака mid_position=(total_digits/2)+1. Далее берётся число mid_value из позиции mid_position в p(i,j) и вычисляется ключ key1= $p(i,j)$ ^{mid_value}. Если это число превышает количество пикселей изображения, то берётся его остаток от деления на количество пикселей key1=key1$_{\mathrm{mod}T}$. Далее берётся десятичное значение 4_bit_dec последних 4 бит пикселя номер $p(i,j)$ } и вычисляется

key2=($_{\text{key1mod (4\_bit\_dec)}}$)+1.

В $p(i,j)$ } –й пиксель встраивается 2 бита данных по правилу: если key2-чётное, то применяется прямое встраивание, если нечётное- то обратное. Если key1- чётное то встраиваются два бита сообщения, нечётное- встраиваются комплементарные 2 бита. К полученному изображению применяется обратное преобразование Арнольда.

Частотные методы

Частотные методы работают с коэффициентами в частотной области.

DCT **(англ. Discrete Cosine Transform — Дискретное косинусное преобразование).** Данный метод использует DCT-преобразование для перехода в частотную область и представляет собой LSB в применении к коэффициентам DCT. Поскольку сжатие JPEG так же использует DCT преобразование, то данную методику возможно применить к сжатым

JPEG-изображениям. При использовании формата JPEG, встраивание производится после сжатия с потерями, использующего DCT, но до применения кода Хаффмана для дальнейшего сжатия коэффициентов DCT без потерь.

Встраивание

Исходное изображение-контейнер разделяется на блоки O_i по 8x8 пикселей, к которым применяется DCT: $F_i[a,b]=DCT(O_i[a,b])\}$. Из каждого коэффициента матрицы F_i выделяются наименее значимые биты и заменяются на биты скрываемого сообщения.

Извлечение

Изображение-контейнер разделяется на блоки O_i по 8x8 пикселей, к которым применяется DCT: $F_i[a,b]=DCT(O_i[a,b])\}$. Из каждого коэффициента матрицы F_i выделяются наименее значимые биты и объединяются, восстанавливая скрытое сообщение.

DWT (англ. Discrete Wavelet Transform — Дискретное вейвлет-преобразование)

По своей сути данная методика схожа с основанной на DCT, но вместо DCT-преобразования для перехода в частотную область используется DWT-преобразование. Один из предложенных методов, основанных на DWT-преобразовании предполагает определение областей изображения, содержащих цвет человеческой кожи в пространстве HSV, затем применяется DWT-преобразование и данные встраиваются только в эти области.

Список использованных источников

1. Дебелов В.А., Валеев Т.Ф. Начальный курс компьютерной графики.- Новосибирск, 2011 - [Электронный ресурс] – Режим доступа:http://nsucgcourse.github.io/lectures/DraftCGintroCourse_.pdf
2. Батура В.А., Тропченко А.Ю., Тропченко А.А. Обработка изображений в системе MATLAB: лабораторные работы– СПб: Университет ИТМО, 2019. – 41 с.
3. Гонсалес Р., Вудс Р., Эддинс С. Цифровая обработка изображений в среде MATLAB.- М.: Техносфера, 2006, - 616 с.
4. Грибунин В. Г., Костюков В. Е., Мартынов А. П., Николаев Д. Б., Фомченко В. Н. Стеганографические системы. Критерии и методическое обеспечение: Учебно-методическое пособие / Под редакцией доктора технических наук В. Г. Грибунина. — г. Саров: ФГУП "РФЯЦ-ВНИИЭФ", 2016.— 324 с.

5. Документация Матлаб на русском языке – [Электронный ресурс] – Режим доступа:https://docs.exponenta.ru/documentation-center.html?s_tid=CRUX_lftnav
6. Доля П.Г. Методы обработки изображений. Учебное пособие. Харьковский Национальный Университет, 2013. - [Электронный ресурс] – Режим доступа: http://geometry.karazin.ua/resources/documents/20191108174923_19583744.pdf
7. Кучеренко Н.Л. MATLAB: типы данных, массивы, работа с файлами, интерфейс: Учебно-методическое пособие. – Бишкек: КРСУ, 2011. – 95 с.
8. Приоров, А.Л. Цифровая обработка изображений: учебное пособие/ А.Л. Приоров, И.В. Апальков, В.В. Хрящев; Яросл. гос. ун-т. – Ярославль: ЯрГУ, 2007. – 235 с.
9. Программирование графического интерфейса пользователя . Институт инженерной физики и радиоэлектроники СФУ - – [Электронный ресурс] – Режим доступа:https://studfile.net/preview/6319551/
10. Сейеди С. А., Садыхов Р. Х. Сравнение методов стеганографии в изображениях/ Информатика. 2013. № 1, с. 66-75
11. Создание законченных приложений на основе графического интерфейса пользователя– [Электронный ресурс] – Режим доступа:http://www.butovo.com/~zss/matlab/4/1.htm
12. LSB based Image steganography using MATLAB – [Электронный ресурс] – Режим доступа:https://www.geeksforgeeks.org/lsb-based-image-steganography-using-matlab/

Содержание

Учебное пособие

Чернецова Елена Анатольевна,
кандидат технических наук, доцент кафедры
информационных технологий и систем безопасности

Визуализация инженерных расчетов в Матлаб

Печатается в авторской редакции.

Подписано к публикации 02.02.2024. Формат 60×90 $^{1}/_{8}$.
Гарнитура Times New Roman.Усл. печ. л. 11,5. Заказ №1462.
РГГМУ, 192007, Санкт-Петербург, Воронежская ул., 79.