

# МATLAB в научных исследованиях

Н. Ю. Золотых<sup>1</sup>

5 ноября 2004 г.

<sup>1</sup>Нижегородский государственный университет им.Н.И.Лобачевского, факультет вычислительной математики и кибернетики

# Оглавление

<b>Глава 1. Введение</b>	<b>4</b>
<b>Глава 2. Простейшие команды</b>	<b>5</b>
2.1. Числа и константы . . . . .	5
2.2. Основные функции для работы с матрицами . . . . .	8
2.3. Массивы символов . . . . .	12
2.4. Форматированный вывод . . . . .	12
2.5. Справка и документация . . . . .	15
2.5.1. Команда help . . . . .	15
2.5.2. Команда lookfor . . . . .	16
2.5.3. Справочный навигатор . . . . .	17
2.6. Среда МАТЛАВ . . . . .	17
2.6.1. Рабочее пространство . . . . .	17
2.6.2. Сохранение и загрузка переменных . . . . .	18
2.6.3. Команды dir, type, delete, cd . . . . .	20
2.6.4. Дневник работы . . . . .	20
2.6.5. Запуск внешних программ . . . . .	21
2.6.6. Редактор командной строки . . . . .	21
2.7. m-файлы . . . . .	21
<b>Глава 3. Простейшая графика</b>	<b>22</b>
3.1. Функция plot . . . . .	22
3.1.1. Команда figure . . . . .	23
3.1.2. Несколько кривых на графике . . . . .	23
3.1.3. Графики в полярных координатах . . . . .	26
3.1.4. Графики многозначных функций . . . . .	27
3.1.5. Стил и цвет линий . . . . .	27
3.1.6. Команды axis и grid . . . . .	29
3.2. Трехмерная графика . . . . .	29

3.2.1. Команда meshgrid . . . . .	29
3.2.2. Команда mesh . . . . .	30
3.3. Примеры . . . . .	33
3.3.1. Тор . . . . .	33
3.3.2. Улитка . . . . .	35
3.3.3. Лист Мебиуса . . . . .	36
3.3.4. Бутылка Клейна . . . . .	36
3.4. Линии уровня . . . . .	38
3.5. Make it easy . . . . .	38
3.6. Изображения (images) . . . . .	39
3.7. Графический способ решения уравнений . . . . .	39
3.7.1. Команда hold on . . . . .	41
3.8. Относительный и абсолютный способ задания цвета . . . .	41
3.9. Неявные функции . . . . .	42
<b>Глава 4. Более сложные элементы языка MATLAB</b>	<b>43</b>
4.1. Суммы . . . . .	43
4.2. Произведения . . . . .	44
4.3. Факториал . . . . .	45
4.4. Сочетание произведений и суммирования . . . . .	45
4.5. Интегрирование . . . . .	47
4.5.1. Интегрирование функций с особенностью . . . . .	48
4.5.2. Аналитический учет особенности: . . . . .	48
4.5.3. “Численный” учет особенности: . . . . .	48
4.6. Управляющие конструкции . . . . .	49
4.6.1. Оператор if . . . . .	49
4.6.2. Оператор while . . . . .	49
4.6.3. Оператор for . . . . .	50
4.6.4. Оператор switch . . . . .	50
4.7. Хронометрирование . . . . .	51
4.7.1. Работа вдоль строк и вдоль столбцов . . . . .	53
4.7.2. Вычисление элементарных функций . . . . .	53
4.7.3. Циклы . . . . .	53
4.7.4. Присваивания . . . . .	53
4.7.5. Двумерные и одномерные индексы . . . . .	54
4.8. Типы данных . . . . .	55
4.8.1. Многомерные массивы . . . . .	55
4.8.2. Массивы структур . . . . .	55
4.8.3. Массивы ячеек . . . . .	56

<b>Глава 5. Многочлены</b>	<b>58</b>
5.1. Основные команды . . . . .	58
5.2. Примеры . . . . .	60
5.3. Сумма многочленов . . . . .	61
5.4. Пример численной неустойчивости . . . . .	62
5.5. Многочлены Лежандра . . . . .	64
<b>Глава 6. Матрицы и линейная алгебра</b>	<b>69</b>
6.1. Простые команды . . . . .	69
6.2. Решение систем линейных уравнений . . . . .	70
6.3. Спектр. Число обусловленности . . . . .	74
6.4. Нормы векторов и матриц . . . . .	74
6.5. Матрицы Гильберта . . . . .	74
6.6. Экспериментальное исследование числа обусловленности	76
<b>Глава 7. Интерполяция и аппроксимация данных</b>	<b>78</b>
7.1. Интерполяция . . . . .	78
7.2. Кусочно-полиномиальные функции . . . . .	82
7.3. Кусочно-полиномиальная интерполяция . . . . .	84
7.4. Кусочно-линейная интерполяция . . . . .	88
7.5. Эрмитов кубический интерполянт . . . . .	89
7.6. Сплайны . . . . .	92
7.7. Кривые Безье . . . . .	95
7.8. В-сплайны . . . . .	99
7.9. Метод наименьших квадратов . . . . .	103
<b>Глава 8. Анализ данных ?</b>	<b>107</b>
8.1. Преобразование Фурье . . . . .	107
<b>Глава 9. Программирование</b>	<b>110</b>
9.1. М-файлы . . . . .	110
9.2. Программы-сценарии . . . . .	111
9.3. Программы-функции . . . . .	112
9.3.1. Подфункции . . . . .	113
9.4. Частные функции . . . . .	114
<b>Глава 10. Библиотека символьной математики</b>	<b>115</b>
10.1. Создание символьных переменных . . . . .	115
10.2. Символьно-числовые преобразования . . . . .	116
10.2.1. Создание абстрактных функций . . . . .	116
10.2.2. Вызов функций Maple . . . . .	116
10.2.3. Создание символьных матриц . . . . .	116

10.3. Создание символьных математических функций . . . . .	117
10.4. Математический анализ . . . . .	117
10.4.1. Производные . . . . .	117
10.4.2. Пределы . . . . .	118
10.4.3. Интегрирование . . . . .	119
10.5. Упрощения и подстановки . . . . .	119
10.5.1. Упрощения . . . . .	119
10.5.2. Подстановки . . . . .	120
10.6. Ортогональные многочлены . . . . .	121
<b>Глава 11. Графические команды и функции</b>	<b>123</b>
11.1. Дескрипторная графика . . . . .	123
11.2. Пример: создание графического объекта . . . . .	125
11.3. Текущие графические объекты . . . . .	125
11.4. Определение граней объекта Patch . . . . .	125
11.4.1. Определение граней объекта Patch с помощью за- дания координат XData, YData, ZData . . . . .	126
11.4.2. Определение граней объекта Patch с помощью за- дания массивов Vertices и Faces . . . . .	127
11.5. Стилль ребер и вершин . . . . .	128
11.6. Цветовые характеристики объекта Patch . . . . .	128
11.6.1. Свойство FaceColor . . . . .	128
11.6.2. Свойство EdgeColor . . . . .	129
11.6.3. Свойство MarkerEdgeColor . . . . .	129
11.6.4. Свойство MarkerFaceColor . . . . .	129
11.6.5. Высококачественные реалистичные изображения .	130
11.6.6. Свойство CData . . . . .	130
11.6.7. Свойство FaceVertexCData . . . . .	134
11.7. Освещение . . . . .	136
11.7.1. AmbientStrength . . . . .	136
11.7.2. DiffuseStrength . . . . .	136
11.7.3. SpecularStrength . . . . .	137
11.7.4. SpecularExponent . . . . .	137
11.7.5. SpecularColorReflectance . . . . .	138
11.7.6. Материал . . . . .	138
11.8. Прозрачность . . . . .	140
11.9. Normals . . . . .	140
11.10 3d графика. Примеры . . . . .	140

**Глава 12. Создание графического интерфейса пользователя 153**

12.1. Элементы управления uicontrol . . . . .	153
12.2. Этапы разработки графического интерфейса . . . . .	154
12.3. Начало работы с GUIDE . . . . .	154
12.4. Параметры создаваемого приложения . . . . .	154
12.4.1. Параметры изменения размеров окна . . . . .	154
12.4.2. Параметры, регулирующие взаимодействие с ко- мандной строкой . . . . .	155
12.5. Callback-функции . . . . .	156
12.5.1. Callback-функции всех типов графических объектов	156
12.5.2. Callback-функции объекта Figure . . . . .	156
12.5.3. Порядок вызова callback-функций . . . . .	157
12.6. Функция guihandles . . . . .	157
12.7. Функция guidata . . . . .	157
12.8. Примеры . . . . .	158

# Предисловие

## Глава 1

# Введение

История развития системы МАТЛАВ (сокращение от «matrix laboratory») насчитывает более двух десятков лет. «Классический» МАТЛАВ был написан Кливом Моулером (университет Нью-Мехико) в 1977 г. Он представлял из себя интерактивную матричную лабораторию, позволяющую вызывать подпрограммы из пакетов LINPACK и EISPACK. До 1984 г. появлялись новые некоммерческие версии системы. В 1985 г. Моулер, Бангерт и Литтл образовали фирму MathWorks. С этого момента начинают выходить коммерческие версии системы. К настоящему моменту (август 2004 г.) последней является версия МАТЛАВ 7.0 R14. Сейчас МАТЛАВ представляет собой мощный математический пакет со своим языком программирования, гибкими графическими возможностями, средствами сопряжения с другими языками программирования и несколькими десятками пакетов приложений.

Начиная с версии МАТЛАВ 6.0, для матричных вычислений вместо функций из библиотек LINPACK и EISPACK используются подпрограммы из пакета LAPACK. На некоторых классах задач это увеличило производительность в 2–3 раза.



## Простейшие команды

### 2.1. Числа и константы

После запуска системы MATLAB на экране появляется основное окно, содержащее несколько подокон. Одно из них имеет заголовок COMMAND WINDOW — это командное окно, в котором пользователь набирает команды, а MATLAB выдает результаты. Результаты выполнения команд, содержащие графический вывод, выдаются в одно или несколько графических окон. Команды пользователя нужно вводить после приглашения системы, которое выглядит следующим образом:

```
>>
```

Например,

```
>> (76+21-85)*3/4
```

(ввод заканчивается клавишей **Enter**). MATLAB выдаст ответ:

```
ans =
```

```
9
```

Теперь наберем:

```
>> (1+sqrt(5))/2
```

получим:

```
ans =
```

```
1.6180
```

В этом примере мы использовали функцию `sqrt` для нахождения квадратного корня; `ans` — это специальная переменная, в которую всегда засылается результат последней команды. Эту переменную можно использовать в следующей команде. Например,

```
>> 1/ans
```

Матлаб выдаст:

```
0.6180
```

Пользователь может создавать свои переменные. Например, команда

```
>> e=2+1/2+1/6+1/24+1/120+1/720
e =
```

```
2.7181
```

создает переменную с именем `e` и значением `2.7181`. Теперь переменную `e` можно использовать. Например,

```
>> err=e-exp(1)
```

Получим:

```
err =
```

```
-2.2627e-4
```

Функция `exp` вычисляет экспоненту  $e^x$ . Запись `-2.2627e-4` — это представление числа в форме с плавающей запятой. Его нужно понимать следующим образом:

$$-2.2627e-4 = -2.2627 \cdot 10^{-4} = -0.00022627.$$

Перед тем как использовать переменную, ее нужно инициализировать, т. е. присвоить ей некоторое значение (так и было в двух предыдущих примерах с переменными `e` и `err`). Использовать неинициализированные переменные запрещено. Например, если `p` и/или `k` ранее не встречались в левой части присваивания, то следующая команда

```
>> x=-p+sqrt(p^2-q)
```

приведет к сообщению об ошибке.

До сих пор знаком окончания команды являлся символ конца строки: ввод команды просто заканчивался клавишей **Enter**. В результате мы всегда получали эхо (отклик). В конце команды, перед тем, как ввести **Enter**, можно поставить знак `;` (точка с запятой). В этом случае отклика системы мы не получим. Например, после того как мы введем

```
>> e=2+1/2+1/6+1/24+1/120+1/720;
```

переменной `e` будет присвоено вычисленное значение 2.7181 и мы сразу получим новое приглашение:

```
>>
```

Далее в примерах, как правило, знак приглашения мы печатать не будем.

В одной строке можно набирать несколько команд. Их нужно отделять либо символом `','` (запятая), либо символом `','` (точка с запятой). В первом случае отклик будет, во втором — нет.

Именами переменных могут быть любые последовательности латинских букв (в любом регистре), цифр и знаков подчеркивания. Первым символом в имени может быть либо буква, либо символ подчеркивания. Ограничений на длину имени нет, но при сравнении имен роль играют только первые 31 символ. Кроме того, важен регистр: например, переменные `Err` и `err` — разные.

Кроме `ans` в системе МАТЛАВ есть другие встроенные переменные. Вот некоторые из них<sup>1</sup>:

<code>pi</code>	$\pi = 3.14159\dots$
<code>i, j</code>	мнимая единица ( $i^2 = -1$ )
<code>realmax</code>	$1.7977 \cdot 10^{308}$
<code>realmin</code>	$2.2251 \cdot 10^{-308}$
<code>eps</code>	$2.2204 \cdot 10^{-16}$
<code>inf</code>	$+\infty$
<code>-inf</code>	$-\infty$
<code>NaN</code>	not a number

МАТЛАВ поддерживает стандарт IEEE 754 арифметики вещественных чисел с плавающей запятой. Под действительное число отводится 8 байт: 16 десятичных цифр мантиссы, показатель в диапазоне  $\pm 308$ . Максимальное представимое вещественное число хранится в переменной `realmax`, минимальное положительное вещественное число — в переменной `realmin`. Константа `eps` (*машинное эpsilon*) — это минимальное положительное число  $\varepsilon_M$ , такое, что в машинной арифметике

$$1 + \varepsilon_M > 1.$$

Константы `inf`, `-inf` в IEEE арифметике служат для представления  $\pm\infty$ . Они получаются при переполнениях. Константа `NaN` появляется в выражениях вида  $0/0$ ,  $\infty/\infty$ ,  $\infty-\infty$  и им подобным. Константа `pi` — это

---

<sup>1</sup> в таблице приводятся приближенные значения констант для машины PC

число  $\pi$ . Комплексные числа на PC представляются 16 байтами. Константы  $i$ ,  $j$  представляют мнимую единицу. Поэтому, например  $1+2*i$  — это комплексное число  $1 + 2i$ .

Все приведенные выше константы могут появляться в правой части присваивания. После этого они теряют свое первоначальное значение. Например, переменные  $i$ ,  $j$  часто используются как счетчики в циклах. После этого  $1+2*i$  будет скорее всего уже не комплексным числом  $1+2i$ , а чем-то другим. Однако запись  $1+2i$  (мы опустили знак умножения) всегда означает комплексное число  $1 + 2i$ . Вернуть первоначальное значение предопределенных констант можно командой `clear`. Например,

```
clear i
```

возвращает значение константе  $i$  как мнимой единице.

Мы уже встречались с функциями `sqrt` и `exp`. В систему MATLAB встроено большое число других стандартных математических функций. Приведем небольшой список некоторых из них (список всех элементарных математических функций, доступных в системе MATLAB, можно получить набрав команду `help elfun`):

<code>sin, cos, tan</code>	тригонометрические функции
<code>acos, asin, atan</code>	обратные тригонометрические функции
<code>exp, log</code>	экспонента и натуральный логарифм
<code>sqrt</code>	квадратный корень
<code>round</code>	округление до ближайшего целого
<code>fix</code>	округление с отбрасыванием дробной части
<code>abs</code>	модуль вещественного или комплексного числа
<code>angle</code>	аргумент комплексного числа
<code>real, imag</code>	вещественная и мнимая части
<code>conj</code>	комплексное сопряжение

## 2.2. Основные функции для работы с матрицами

Основной объект в системе MATLAB — это матрицы, или массивы. Даже скалярные величины, с которыми мы имели дело в предыдущем разделе, рассматриваются системой как матрицы  $1 \times 1$ .

Вектор (одномерный массив) может представлять собой строку или столбец, т. е. либо матрицу размера  $1 \times n$ , либо матрицу размера  $m \times 1$ . Чтобы задать вектор, достаточно перечислить его элементы, заключая их в квадратные скобки. Элементы векторов-строк разделяются символами `','` (запятая) или `' '` (пробел). Элементы векторов-столбцов разделяются символом `','` (точка с запятой) или символом перехода на новую строку, который нужно ввести клавишей `Enter`. Например, команда

```
a=[1 2 3 4]
```

задает строку  $a = (1, 2, 3, 4)$ , а команда

```
b=[1;2;3;4]
```

задает столбец

$$a = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix}.$$

Векторы, состоящие из последовательных членов арифметической прогрессии, легко задавать с помощью команды `‘:’`. Команда `a:h:b` определяет вектор

$$(a, a + h, a + 2h, \dots, b).$$

Команда `a:b` определяет отрезок арифметической прогрессии с шагом 1.

Чтобы задать матрицу (двумерный массив), достаточно перечислить ее элементы построчно, разделяя элементы в одной строке пробелами или запятыми, а сами строки — точкой с запятой или символом перехода на новую строку. Например, команда

```
A = [1 2; 3 4]
```

определяет матрицу

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}.$$

Количество элементов в одной строке должно быть одинаковым, иначе MATLAB выдаст сообщение об ошибке.

Для генерирования матриц полезны следующие простые функции:

<code>zeros(m,n)</code>	нулевая матрица
<code>ones(m,n)</code>	матрица, состоящая из одних единиц
<code>eye(m,n)</code>	матрица с 1 на диагонали и 0 вне диагонали
<code>rand(m,n)</code>	матрица со случайными элементами, равномерно распределенными на отрезке $[0, 1]$
<code>randn(m,n)</code>	матрица со случайными элементами, распределенными по нормальному закону с мат. ожиданием 0 и ср. кв. отклонением 1

В приведенной таблице параметры `m`, `n` определяют число строк и число столбцов матрицы. Каждую из предыдущих функций можно вызвать с

одним параметром — в этом случае генерируется квадратная матрица указанного порядка с соответствующим свойством. Например, команда `eye(m)` генерирует единичную матрицу порядка  $m$ .

К матрицам применимы все стандартные математические функции: они применяются покомпонентно к каждому элементу. В отличие от этих функций операции, обозначаемые символами, подобными  $+$ ,  $*$ , выполняются как *матричные* операции. Приведем список операций:

$a+b$	сложение скаляров, векторов или матриц
$a-b$	вычитание скаляров, векторов или матриц
$a*b$	умножение скаляров; матричное умножение
$a.*b$	покомпонентное умножение элементов матриц
$a.^b$	возведение скаляра или матрицы в степень
$a.^b$	возведение каждого элемента матрицы в степень
$a./b$	деление скаляров; правое деление матриц, т. е. $a \cdot b^{-1}$
$a ./b$	покомпонентное деление элементов матриц
$a \backslash b$	левое деление матриц, т. е. $a^{-1} \cdot b$
$a'$	транспонирование матрицы

Допустимо использование операций  $+$ ,  $-$ ,  $*$ , когда один из операндов — матрица, а другой операнд — скаляр. Это приводит к покомпонентному выполнению операции. Если размеры операндов (матриц, участвующих в операции) не согласованы, то выдается сообщение об ошибке.

Нумерация строк и столбцов в матрицах начинается с 1. Чтобы получить доступ к элементу матрицы  $A$ , стоящему в  $i$ -ой строке,  $j$ -ом столбце, достаточно ввести  $A(i, j)$ . Чтобы получить доступ к  $k$ -ой компоненте вектора  $a$  достаточно ввести  $a(k)$ . Команда  $A(k)$  работает также и для матриц: MATLAB ищет  $k$ -ый элемент матрицы  $A$ , предполагая, что элементы нумеруются по столбцам.

Что произойдет, если в команде есть ссылка на несуществующий элемент матрицы? Например, матрица  $A$  имеет размеры  $2 \times 2$  и происходит обращение к элементу  $A(3, 5)$ . Все зависит от того, в какой части от знака присваивания расположена ссылка на элемент  $A(3, 5)$ . Если  $A(3, 5)$  находится где-то в выражении справа от знака присваивания, это приведет к сообщению об ошибке. Если  $A(3, 5)$  стоит слева, то размеры матрицы автоматически переопределяются до  $3 \times 5$ , при этом новым элементам матрицы присваиваются нулевые значения, а элементу  $A(3, 5)$  — вычисленное значение.

Для ссылки к последнему столбцу или строке можно использовать ключевое слово `end`. Например,  $A(\text{end}, k)$  — это элемент матрицы  $A$ , стоящий в последней строке и  $k$ -ом столбце.

Для ссылки ко всем строкам или столбцам матрицы используется команда `:`. Например, `A(:,k)` — это  $k$ -ый столбец, а `A(k,:)` —  $k$ -ая строка матрицы  $A$ . Подобные выражения могут встречаться и в левой части присваивания. Например, команда

```
A(k,:)=3
```

присваивает всем элементам  $k$ -ой строки значение 3.

Результатом операции `A(:)` является длинный столбец, составленный из столбцов матрицы  $A$ . Таким образом, `A(k)` — это  $k$ -ый элемент вектора `A(:)`.

Функция `size(A)` возвращает двухкомпонентный вектор, содержащий число строк и число столбцов матрицы  $A$ . Функция `size(A,1)` возвращает число строк, функция `size(A,2)` — число столбцов. `length(A)` — это максимум из этих двух чисел, поэтому для векторов (в этом случае одна из размерностей равна 1) — это число компонент в них.

В системе МАТЛАВ легко реализуются блочные операции над матрицами. Команды `C=[A B]`, `C=[A;B]` формируют из матриц  $A$  и  $B$  блочные матрицы соответственно

$$C = (A|B) \quad \text{и} \quad C = \begin{pmatrix} A \\ B \end{pmatrix}.$$

Пусть  $u, v$  — векторы, в которых записаны номера некоторых строк и столбцов соответственно (быть может, с повторениями) матрицы  $A$ . Тогда `A(u,v)` — это матрица, составленная из элементов исходной матрицы, стоящих на пересечении строк с номерами  $u$  и столбцов с номерами  $v$ .

Система МАТЛАВ поддерживает работу с пустыми матрицами, т.е. матрицами, в которых число строк или/и число столбцов равно нулю. Один из способов задать такую матрицу — воспользоваться функцией типа `zeros`. Например,

```
A=zeros(0,5)
```

Определить матрицу размера  $0 \times 0$  можно с помощью операции `[]`.

Операция `[]` помогает также при удалении строк или столбцов матриц. Команда

```
A(i,:)=[]
```

удаляет  $i$ -ую строку, а команда

```
A(:,j)=[]
```

удаляет  $j$ -ый столбец матрицы  $A$ .

### 2.3. Массивы символов

МАТЛАВ позволяет работать со строковыми значениями. Чтобы определить такое значение, достаточно заключить строку символов в кавычки. Например,

```
s='Isaac Newton'
```

Строковые значения рассматриваются системой как массивы. В приведенном примере `s` — это вектор-строка из 12 элементов-символов. (никакого завершающего нулевого символа нет).

Символы можно объединять в двумерные массивы. Это позволяет хранить набор строковых значений одинаковой длины. Например, после ввода команды

```
S=['Isaac Newton '
   'Blaise Pascal']
```

МАТЛАВ создает следующий двумерный массив  $2 \times 13$ :

$$\begin{pmatrix} \text{'I'} & \text{'s'} & \text{'a'} & \text{'a'} & \text{'c'} & \text{' '}& \text{'N'} & \text{'e'} & \text{'w'} & \text{'t'} & \text{'o'} & \text{'n'} & \text{' '}& \\ \text{'B'} & \text{'l'} & \text{'a'} & \text{'i'} & \text{'s'} & \text{'e'} & \text{' '}& \text{'P'} & \text{'a'} & \text{'s'} & \text{'c'} & \text{'a'} & \text{'l'} & \end{pmatrix}$$

В данном примере мы специально добавили к имени Ньютона один пробел, чтобы уравнять число элементов в каждой строке массива. Если этого не сделать, МАТЛАВ выдаст сообщение об ошибке.

Команда `S(1,:)` в данном примере возвращает строку `'Isaac Newton '`, команда `S(2,:)` — строку `'Blaise Pascal'`. Вообще, к массивам символов применимо большое число команд из предыдущего раздела. Так, например, доступны блочные операции, команды выделения подмассивов и др. В примере

```
a = 'Matrix';
b = 'Laboratory';
c = [a(1:3) b(1:3)]
```

переменной `c` будет присвоено символьное значение `'\MATLAB.'`

Внутри системы МАТЛАВ символы представлены в формате UNICODE. Для хранения одного символа используется 2 байта.

### 2.4. Форматированный вывод

МАТЛАВ предоставляет широкие возможности для управления форматом вывода числовых значений в командном окне. По умолчанию используется формат `short`. В нем используются следующие правила:



- Если *все* элементы массива — целые числа не более чем из 9 цифр каждое, то они выводятся как есть.
- Если *все* элементы массива по абсолютной величине меньше 1000 и не меньше 0.0001, то они выводятся как есть.
- В остальных случаях MATLAB выводит элементы массива с использованием общего множителя (исключение составляют скалярные величины). Элементы выводятся в формате  $\pm d.d\text{d}d\text{e}s\text{p}\text{p}\text{p}$ , где  $d$  — цифры мантиссы,  $p$  — цифры показателя. Если число не ноль, то старшая цифра перед десятичной точкой не равна нулю. Если число отрицательное, то впереди, разумеется, ставится знак  $-$ .

Например, после ввода команды

```
1/7
```

MATLAB напечатает

```
0.1429
```

Управлять форматом вывода можно либо с помощью пункта меню **File | Preferences | Command Window | Numeric Format** или с помощью команды **format** с параметрами. Действие этой команды распространяется для всех последующих выдач, пока не будет введена новая команда **format** с другим параметром. Приведем список некоторых из возможных параметров:

<code>format</code>	по умолчанию, то же, что и <code>short</code> ;
<code>format short</code>	формат представления чисел с фикс. точкой: 5 значащих цифр;
<code>format long</code>	формат представления чисел с фикс. точкой: 14 цифр после запятой;
<code>format short e</code>	формат представления чисел с плав. точкой: 5 значащих цифр;
<code>format long e</code>	формат представления чисел с плав. точкой: 15 значащих цифр;
<code>format rat</code>	аппроксимация чисел рациональной дробью.

Подчеркнем, что команда **format** влияет только на вывод числовых данных и ни коим образом не влияет на то, как хранятся эти данные. Это относится, конечно, и к формату **rat**: всякий раз, когда нужно

вывести числовое значение, МАТЛАВ находит к нему наилучшее рациональное приближение такое, что  $\text{abs}(N./D - X) \leq \text{tol} * \text{abs}(X)$ , где  $\text{tol} = 1.e - 6 * \text{norm}(X(:), 1)$ , которое и выводится на экран.

Рассмотрим пример:

```
>> format compact  
>> 0.3
```

```
ans =
```

```
3/10
```

```
>> 0.33
```

```
ans =
```

```
33/100
```

```
>> 0.333
```

```
ans =
```

```
333/1000
```

```
>> 0.3333
```

```
ans =
```

```
3333/10000
```

```
>> 0.33333
```

```
ans =
```

```
33333/100000
```

```
>> 0.333333
```

```
ans =
```

```
333333/1000000
```

```
>> 0.3333333
```

```
ans =
```

```
1/3
```

## 2.5. Справка и документация

Приведем возможные источники получения справочной информации по системе MATLAB :

- команда `help`,
- команда `lookfor`,
- справочный навигатор вместе с описанием команд и руководством пользователя,
- сайт MathWorks [www.mathworks.com/products/matlab/](http://www.mathworks.com/products/matlab/),
- сайт консультационного центра MATLAB [www.matlab.ru](http://www.matlab.ru) (на русском языке),
- другая информация, доступная в Интернете, например, по адресу <http://directory.google.com/Top/Science/Math/Software/MATLAB>.

### 2.5.1. Команда `help`

Команда

```
help
```

без параметров выдает список основных разделов встроенной справки. Чтобы получить справку по какому-либо разделу, нужно набрать

```
help имя_раздела
```

На экран будет выдан список всех функций, относящихся к этому разделу. Получить справку по той или иной функции можно с помощью команды

```
help имя_функции
```

Например, набрав команду

```
help inv
```

вы получите справку по функции `inv`:

```
INV      Matrix inverse.  
INV(X) is the inverse of  
the square matrix X.  
A warning message ...
```

```
See also SLASH, PINV ...
```

```
Overloaded methods ...
```

Заметим, что в справке командного окна все функции набраны в верхнем регистре. Это сделано лишь для того, чтобы выделить их в тексте. В действительности, имена всех встроенных функций системы МАТЛАВ в нижнем регистре.

### 2.5.2. Команда `lookfor`

Команда `help` полезна лишь в случае, когда мы знаем точное имя раздела или функции. Если таковые нам не известны, можно воспользоваться командой

```
lookfor ключевое_слово
```

Например, мы ищем информацию по функции, которая находит обратную матрицу. Набираем:

```
lookfor inverse
```

Получаем список всех функций, в справке к которым в первой строке содержится слово `inverse`:

```
INVHILB Inverse Hilbert matrix.  
ACOS    Inverse cosine.  
ACOSH   Inverse hyperbolic cosine.  
...  
INV     Matrix inverse.  
...
```

Команда

```
lookfor -all inverse
```

ищет слово `inverse` во всех строках справки.

### 2.5.3. Справочный навигатор

На диске в форматах html и pdf хранится огромный массив справочной информации, а также руководства по ядру системы и пакетам расширений. Браузер для справочных страниц в формате html можно запустить командой

```
helpdesk
```

Если имя команды уже известно используйте команду

```
doc имя_команды_или_функции
```

Команда

```
helpwin
```

предоставляет ту же информацию, что и команда help, но обеспечивает более удобный интерфейс.

## 2.6. Среда MATLAB

В этом разделе мы рассмотрим следующие аспекты работы в системе MATLAB :

- команды who, whos, clear, clear all,
- команды save, load,
- команды dir, type, delete, cd,
- команда diary,
- запуск внешних программ.
- редактор командной строки.

### 2.6.1. Рабочее пространство

Значения всех создаваемых в командном окне переменных хранятся в отведенной области оперативной памяти, которую мы будем называть *основным рабочим пространством*. Помимо основного рабочего пространства MATLAB создает и в нужное время удаляет рабочие пространства вызываемых функций. В этих рабочих пространствах размещаются внутренние переменные функций.

Имена всех переменных, находящихся в настоящий момент в основном рабочем пространстве отображаются в подокне WORKSPACE. Список имен переменных можно также получить командой `who`, а список переменных с их значениями — командой `whos`.

Команда

```
clear список_переменных
```

исключает (стирает) указанные переменные из рабочего пространства. При этом их значения теряются. Команда

```
clear all
```

стирает все переменные рабочего пространства.

Рассмотрим пример:

```
>> a=rand(100);
>> b=rand(100);
>> c=a*b;
>> who
```

Your variables are:

```
a    b    c
```

```
>> clear a b
>> whos
```

Name	Size	Bytes	Class
c	100x100	80000	double array

```
Grand total is 10000 elements using 80000 bytes
```

### 2.6.2. Сохранение и загрузка переменных

После выхода из системы МАТЛАВ все переменные уничтожаются и поэтому не доступны в следующий сеанс работы. Запомнить на диске все переменные основного рабочего пространства можно с помощью команды

```
save имя_файла
```

При этом все переменные сохраняются в бинарном формате в указанном файле. По умолчанию расширение этого файла — `mat`, поэтому файлы такого формата называются `mat`-файлами.

Чтобы запомнить не все, а только часть переменных из основного рабочего пространства, воспользуйтесь командой

```
save имя_файла список_переменных
```

Прочитать mat-файл можно с помощью команды

```
load имя_файла
```

которая загружает все переменные, описанные в файле, или с помощью команды

```
load имя_файла список_переменных
```

которая загружает из файла лишь указанные переменные.

Рассмотрим пример:

```
>> x=(0:4)';  
>> W=[x.^0 x.^1 x.^2 x.^3];  
>> save Wndrmd x W  
>> clear all  
>> load Wndrmd  
>> who
```

Your variables are:

```
W x
```

С помощью команд

```
save имя_переменной -ascii
```

можно сохранить значение переменной в одноименном *текстовом* файле. В данном файле в текстовом формате построчно будут записаны элементы матрицы. Такие файлы можно готовить во внешнем редакторе. Не зависимо от того, как был создан такой файл, его можно прочитать командой

```
load имя_файла -ascii
```

Значения из файла будут загружены в переменную, имя которой совпадает с именем файла (без расширения).

### 2.6.3. Команды `dir`, `type`, `delete`, `cd`

На панели управления в основном окне MATLAB расположено выпадающее меню `CURRENT DIRECTORY`, в котором можно выбрать текущий каталог.

Команда

`dir`

выводит на экран список файлов текущего каталога.

Команда

`type имя_файла`

выводит на экран распечатку указанного файла, расположенного в текущем каталоге.

Команда

`delete имя_файла`

удаляет указанный файл, расположенный в текущем каталоге.

Команда

`cd новый_каталог`

изменяет текущий каталог. То же действие можно проделать, воспользовавшись выпадающим меню в подокне `CURRENT DIRECTORY` или меню на панели управления.

### 2.6.4. Дневник работы

Чтобы записать в файл все, что отображается в командном окне: и ваши команды и ответы системы, — воспользуйтесь командой `diary`. Команда

`diary имя_файла`

открывает дневник, т.е. указывает системе, что все, что появится после этой команды на экране до следующей команды `diary` будет записано в упомянутый текстовый файл. Прерывает запись в дневник команда открытия нового дневника или команда

`diary off`



### 2.6.5. Запуск внешних программ

Запустить на выполнение любую команду ДОС или любую внешнюю программу, находящуюся в текущем каталоге, можно набрав

`!имя_команды`

### 2.6.6. Редактор командной строки

Быстрый способ повторить одну из предыдущих строк системы МАТЛАВ — это нажимать на клавишу ‘стрелка вверх’, пока не появится нужная строка. Если вы хотите получить строку сходную с той, что была введена ранее, воспользуйтесь тем же приемом, чтобы вызвать ее, а затем отредактируйте, пользуясь клавишами со стрелками вправо и влево и клавишей Delete.

Быстрый способ вызвать строку с известным началом — набрать это начало, а дальше, пользуясь клавишами со стрелками вверх и вниз, выбрать нужную строку из строк с таким началом.

## 2.7. m-файлы

## Глава 3

# Простейшая графика

В этой главе мы рассмотрим несколько простейших функций с графическим выводом. Весь графический вывод в системе МАТЛАВ поступает в одно или несколько графических окон. Обычно пользователь не должен беспокоиться об их открытии и закрытии: графические функции высокого уровня, о которых пойдет речь в настоящей главе, обычно ведут себя достаточно разумным образом.

### 3.1. Функция `plot`

Если `x` и `y` — два вектора одинаковой длины, то функция

```
plot(x,y)
```

в графическом окне строит ломаную по точкам с абсциссами, записанными в `x`, и ординатами, записанными в `y`. Масштаб по обеим осям выбирается автоматически, так, чтобы ломаная целиком убиралась на графике. Если до выполнения этой команды ни одно графическое окно открыто не было, то такое окно открывается автоматически. В противном случае вывод будет происходить в последнее (текущее) графическое окно и по умолчанию будет стирать старое изображение

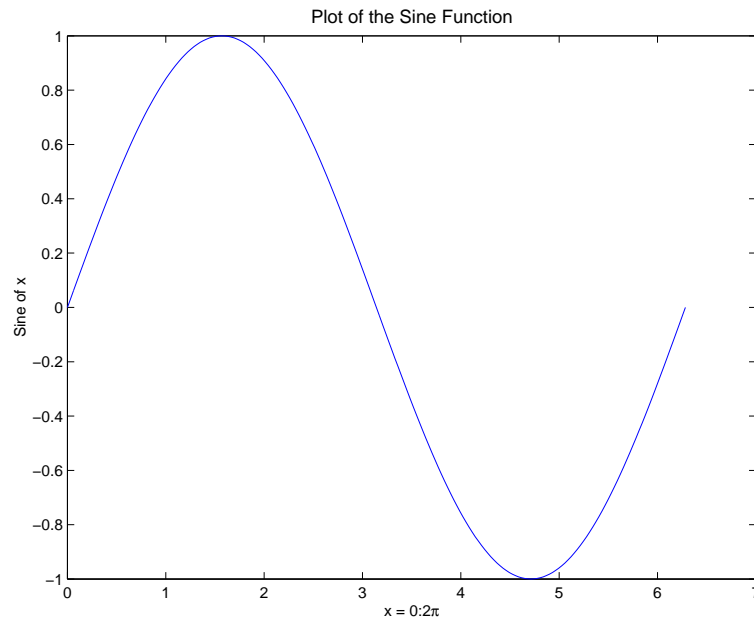
С помощью функции `plot` легко строятся графики функции. Например:

```
x = 0:pi/100:2*pi;  
y = sin(x);  
plot(x,y)
```

Функции `xlabel`, `ylabel` добавляют подписи к оси абсцисс и ординат соответственно, а `title` определяет заголовок.

Применим эти функции для нашего примера:

```
xlabel('x = 0:2\pi')  
ylabel('Sine of x')  
title('Plot of the Sine Function')
```

Рис. 3.1. График функции  $\sin(x)$ 

### 3.1.1. Команда `figure`

Команда `figure` создает новое графическое окно и делает его текущим

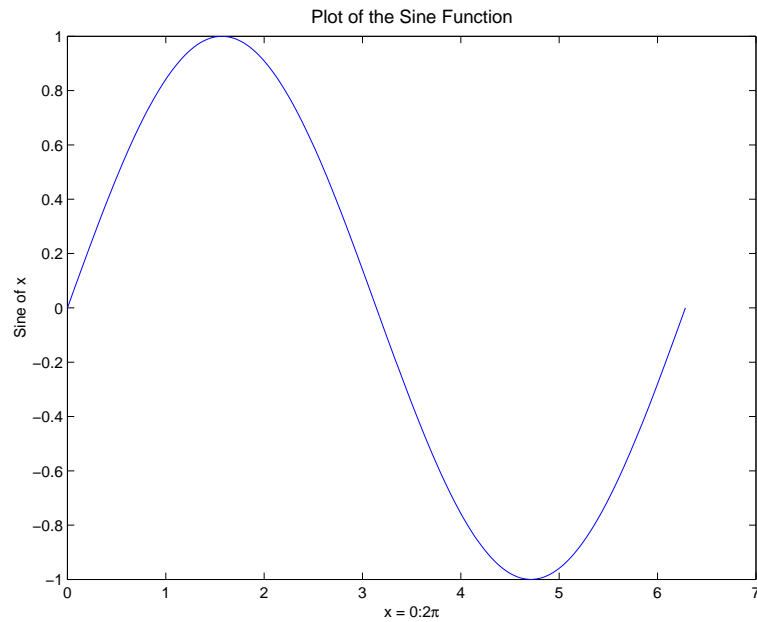
Команда `figure(n)` делает текущим окно с номером `n`

### 3.1.2. Несколько кривых на графике

Каждая новая функция `plot` стирает старое изображение. Для того, чтобы нарисовать несколько графиков, мы можем использовать команду `hold on` (“заморозить”), включающей режим сохранения предыдущего графического результата.

Например, построим графики трех функций (см. рис. 3.1.2):

```
x = -pi:0.1:pi;  
y1 = sin(x);
```

Рис. 3.2. График функции  $\sin(x)$ 

```

y2 = exp(-x.^2);
y3 = .5*atan(x);
plot(x,y1)
hold on
plot(x,y2)
plot(x,y3)
hold off

```

Выйти из режима `hold on` можно с помощью команды `hold off`.

Другой способ вывести несколько графиков в одном окне — применить функцию

```
plot(x1,y1,...,xn,yn)
```

с несколькими парами параметров  $x$ ,  $y$ . Попробуем построить те же графики:

```
plot(x,y1,x,y2,x,y3)
```

Отличие от предыдущего примера в том, что графики нарисованы разными цветами. Ниже мы подробно рассмотрим правила чередования цветов.

Команда `legend` добавляет легенду — пояснение к графикам. У нас три графика, поэтому число параметров функции `legend` должно быть тоже три:

```
legend('sin(x)', 'exp(-x^2)', '0.5 atan(x)')
```

Легенда появится в правом верхнем углу осей координат. В данном случае это не совсем удачно, так как она перекроет часть графиков. Чтобы изменить положение легенды, можно перетащить ее мышкой или воспользоваться функцией `legend` еще с одним параметром:

```
legend('sin(x)', 'exp(-x^2)', '0.5 atan(x)', 2)
```

Получим изображение на рис. 3.1.2. Этим параметром может быть число от 0 до 5. Числа от 1 до 4 соответствуют номеру квадранта, в котором будет размещена легенда: 1 — правый верхний, 2 — левый верхний, 3 — левый нижний, 4 — правый нижний. Если указан 0, то МАТЛАВ сам ищет наиболее удачное место на графике. Если указана 5, то МАТЛАВ размещает легенду снаружи от осей координат.

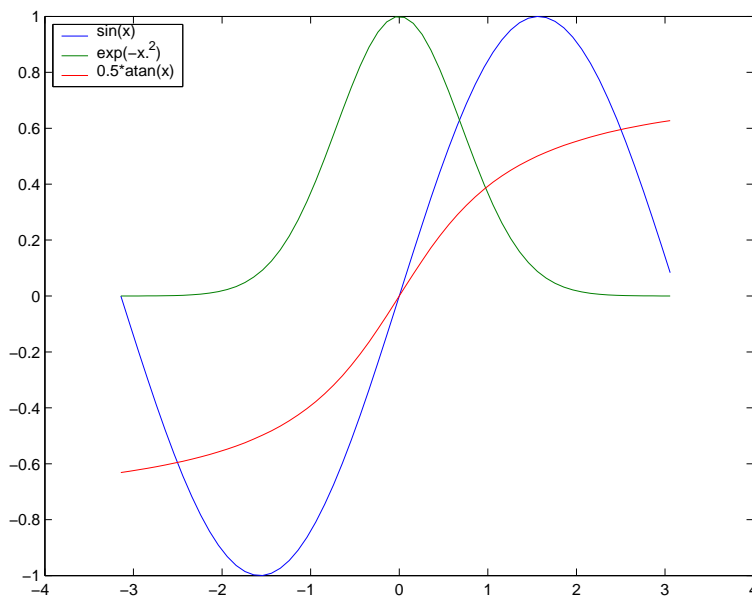


Рис. 3.3. Три графика в одних осях

Если  $x$  — вектор длины  $m$ , а команда

```
plot(x,Y)
```

эквивалентна команде

```
plot(x,Y(:,1),x,Y(:,2),...,x,Y(:,n))
```

Например, графики из предыдущего примера можно получить с помощью команды

```
plot(x,[y1,y2,y3])
```

Нарисуем 8 синусоид с разными амплитудами:

```
a=1:8;  
x=linspace(0,pi,100);  
Y=sin(x)'*a;  
plot(x,Y);
```

и понаблюдаем за порядком чередования цветов. Он следующий:

синий—зеленый—красный—бирюзовый—лиловый—желтый—черный  
blue—green—red—cyan—magenta—yellow—black

Отметим, что этот порядок соответствует следующей последовательности RGB-векторов:

```
[0 0 1]  
[0 1 0]  
[1 0 0]  
[0 1 1]  
[1 0 1]  
[1 1 0]  
[1 1 1]
```

### 3.1.3. Графики в полярных координатах

Графики в полярных координатах выполняются с помощью команды

```
polar(phi,r)
```

Например:

```
phi=linspace(0,2*pi,200);  
polar(phi,sin(4*phi));  
hold on;  
polar(phi,cos(2*phi),'r');  
hold off;
```

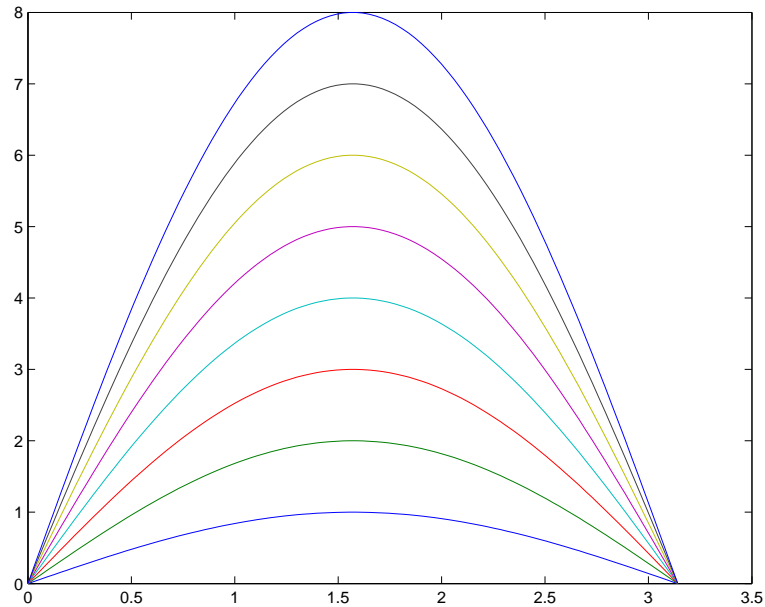


Рис. 3.4. Синусоиды с разными амплитудами

#### 3.1.4. Графики многозначных функций

В системе МАТЛАВ нетрудно получить график обратной функции:

```
x=-3*pi:.01:3*pi;
y=cos(x);
plot(x,y,y,x);
legend('y=cos(x)', 'x=cos(y)');
```

#### 3.1.5. Стиль и цвет линий

Рассмотрим еще один вариант команды `plot`:

```
plot(x,y, стиль)
```

Здесь **стиль** — это строка, состоящая от 1 до 4 символов, обозначающих цвет и стиль линии и тип маркера:

- Цвет: c, m, y, r, g, b, w, and k
- Стиль линии: -, --, :, -. .
- Тип маркера: +, o, \*, x, s, d, ^, v, >, <, p, h

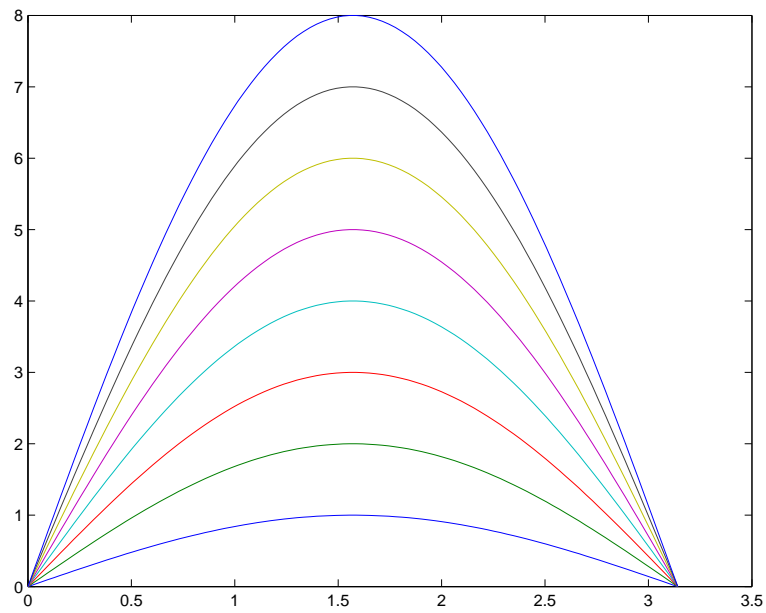


Рис. 3.5. Синусоиды с разными амплитудами

Цвет линии и маркера определяется одной буквой:

Символ	Цвет
'b'	синий
'g'	зеленый
'r'	красный
'c'	бирюзовый
'm'	лиловый
'y'	желтый
'k'	черный

Рассмотрим возможные стили линии:

Символ	Стиль линии
'_'	сплошная линия (по умолчанию)
'_'	штрих-линия
'.'	пунктирная линия
'-.'	штрих-пунктирная линия
'none'	нет линии



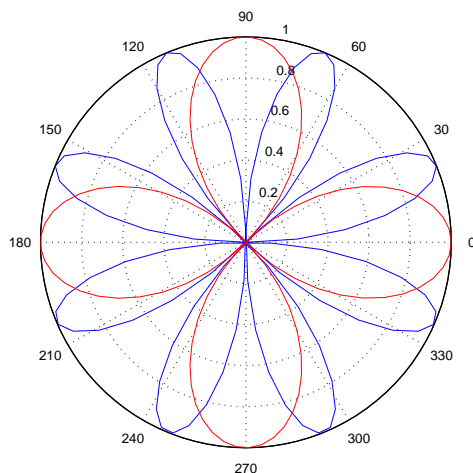


Рис. 3.6. График в полярных координатах

### 3.1.6. Команды axis и grid

Установить пределы выводимой графической информации можно командой `axis([xmin xmax ymin ymax])` или `axis([xmin xmax ymin ymax zmin zmax])`. Командой `axis auto` можно восстановить режим, в котором пределы вычисляются автоматически.

Команда `axis square` устанавливает одинаковые пределы по всем осям. Команда `axis equal` устанавливает одинаковый масштаб.

Команда `grid on` включает, а команда `grid off` выключает режим отображения решетки.

Пространственные кривые

```
t=[1:25:30]; x=t.*cos(t); y=t.*sin(t); z=t; plot3(x,y,z) grid
```

## 3.2. Трехмерная графика

### 3.2.1. Команда meshgrid

Познакомимся с функцией, незаменимой при изображении поверхностей. Пусть  $\mathbf{x}$  и  $\mathbf{y}$  — векторы длины  $n$  и  $m$  соответственно, тогда результатом команды

```
[X,Y]=meshgrid(x,y)
```

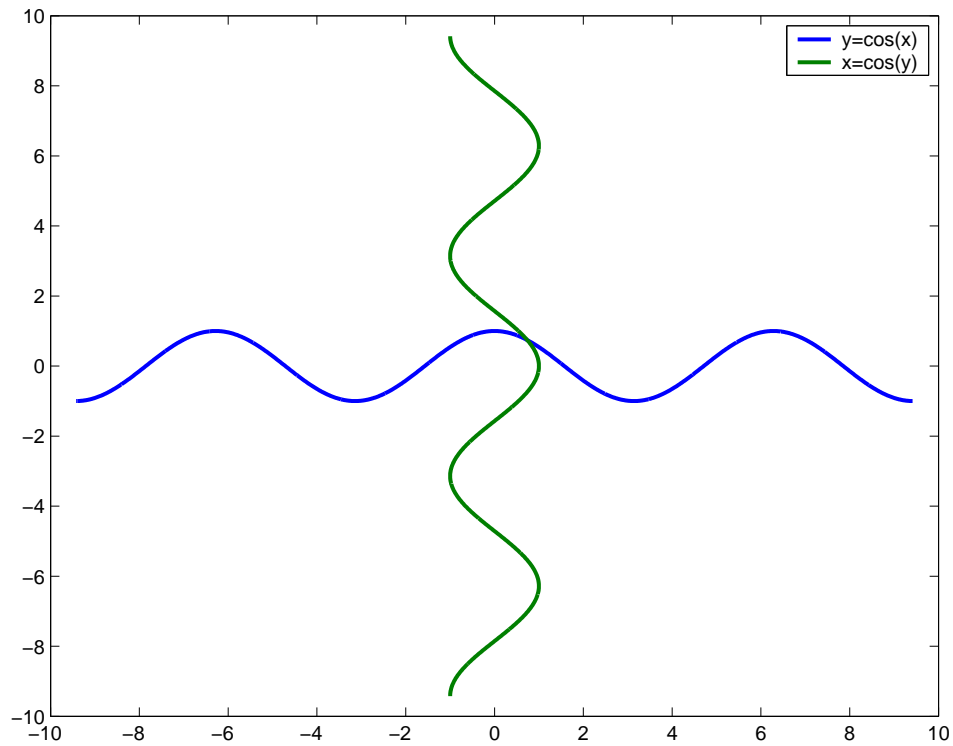


Рис. 3.7. Графики многозначных функций

являются две матрицы размеров  $m \times n$  следующего вида:

$$X = \begin{pmatrix} x(1) & x(2) & \dots & x(n) \\ x(1) & x(2) & \dots & x(n) \\ \dots & \dots & \dots & \dots \\ x(1) & x(2) & \dots & x(n) \end{pmatrix}.$$

$$Y = \begin{pmatrix} y(1) & y(1) & \dots & y(1) \\ y(2) & y(2) & \dots & y(2) \\ \dots & \dots & \dots & \dots \\ y(m) & y(m) & \dots & y(m) \end{pmatrix}.$$

### 3.2.2. Команда mesh

Для рисования графиков функций двух переменных в системе МАТЛАВ имеется несколько функций. Вот некоторые из них:

`mesh(X,Y,Z)`

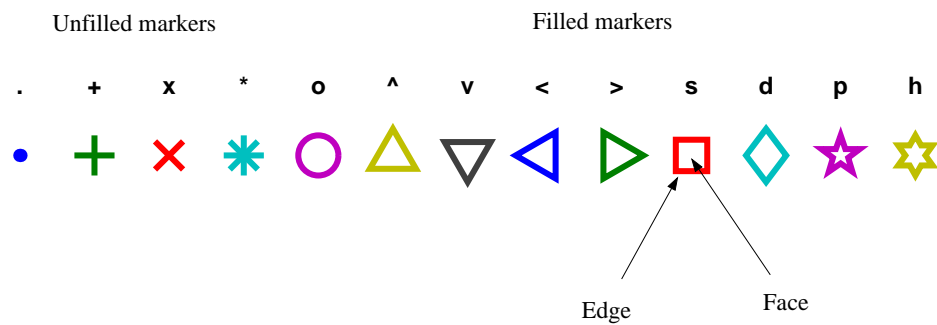


Рис. 3.8. Типы маркеров

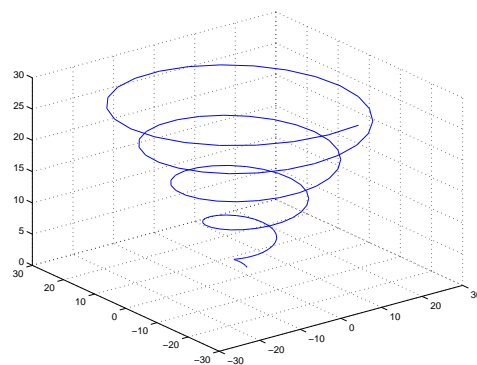


Рис. 3.9. Раскручивающаяся спираль

```
surf(X,Y,Z)
surfl(X,Y,Z)
```

Параметры всех трех приведенных функций имеют одинаковый смысл:  $X$ ,  $Y$ ,  $Z$  — это двумерные массивы, определяющие координаты  $x$ ,  $y$ ,  $z$  поверхности. Функция `mesh` рисует сетчатую (проволочную) поверхность, соединяя прямыми отрезками соседние точки (см. рис. 3.2.2).

Функция `surf` изображает поверхность, а `surfl` — с учетом модели освещения.

```
[X,Y]=meshgrid(-3:.25:3,-3:.25:3);
Z=sin(X).*sin(Y);
mesh(X,Y,Z);
```

```
hidden off
```

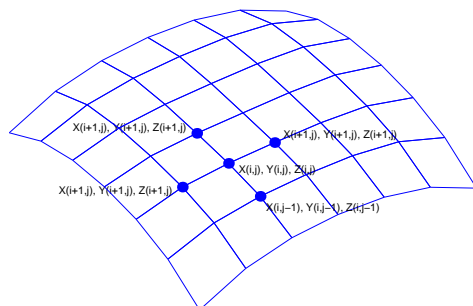


Рис. 3.10. Параметры X, Y, Z функции mesh

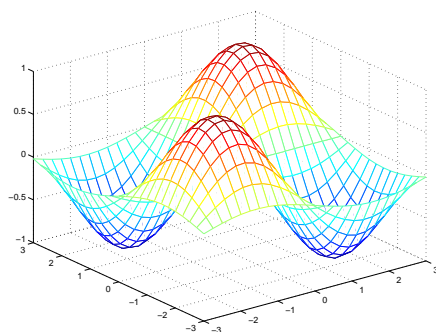


Рис. 3.11. Проволочная модель поверхности

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

```
surf(X,Y,Z)
```

```
colormap (default hsv)
```

```
colorbar
```

```
surfc(X,Y,Z)
```

```
surfl(X,Y,Z)
```

```
colormap
```

```
colormap summer (bone copper gray winter autumn spring hot pink)
```

Тушевка:

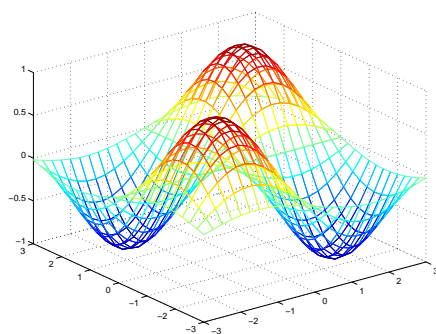


Рис. 3.12. Проволочная модель поверхности

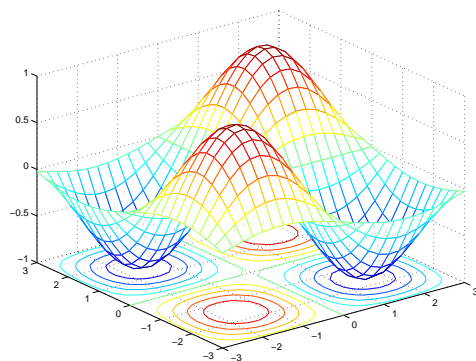


Рис. 3.13. Проволочная модель поверхности

shading faceted (по умолчанию)  
 shading flat --- то же, что и faceted, но ребра не высвечиваются  
 shading interp ---

### 3.3. Примеры

Параметрические поверхности:

#### 3.3.1. Top

R=5;  
 r=2;  
 n=40;

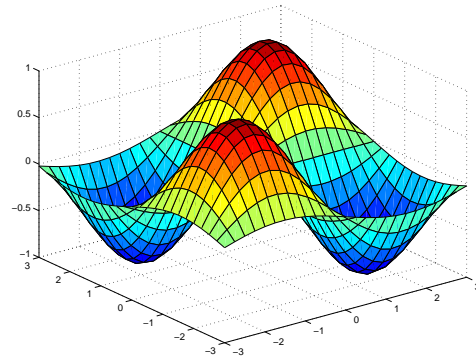


Рис. 3.14. Проволочная модель поверхности

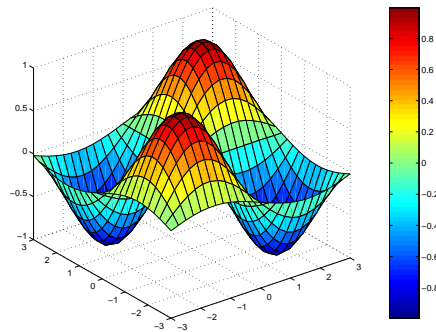


Рис. 3.15. Проволочная модель поверхности

```

m=20;

[U,V]=meshgrid(linspace(-0.1,2*pi,n),linspace(-0.1,2*pi,m));

X=(R+r.*cos(V)).*cos(U);
Y=(R+r.*cos(V)).*sin(U);
Z=r.*sin(V);

surf(X,Y,Z)
axis([-5 5 -5 5 -5 5])
colormap bone

```

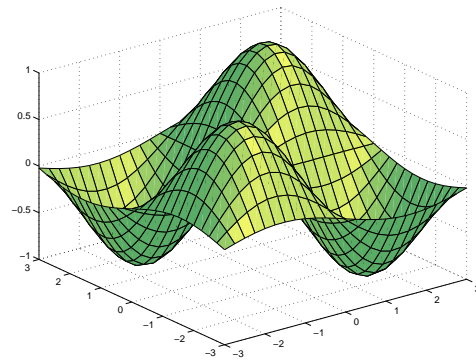


Рис. 3.16. Проволочная модель поверхности

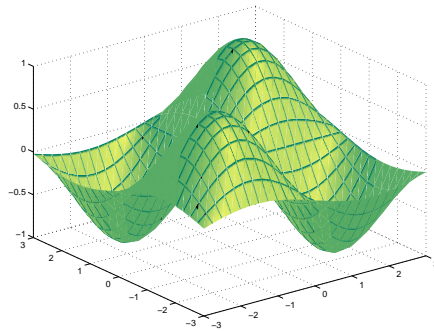


Рис. 3.17. Проволочная модель поверхности

### 3.3.2. Улитка

```
[u,v]=meshgrid(linspace(0,11*pi,500),linspace(-pi,pi,50));
surfl(u.*cos(u).*(cos(v)+5), ...
      u.*sin(u).*(cos(v)+5), ...
      u.*sin(v) - ((u+5)/8*pi).^2 - 20);
view(-5,16);
shading interp;
colormap winter;
axis equal;
axis off;
```

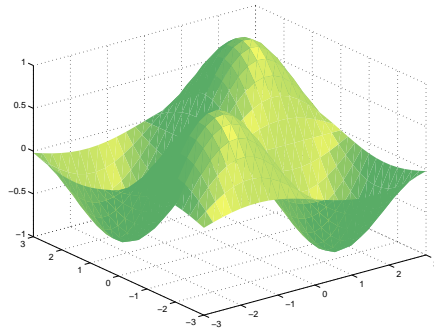


Рис. 3.18. Проволочная модель поверхности

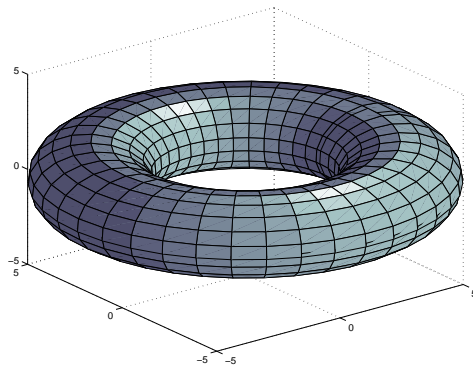


Рис. 3.19. Проволочная модель поверхности

### 3.3.3. Лист Мебиуса

```
[u,v]=meshgrid(linspace(0,2*pi,20),linspace(-.1,.1,10));
mesh(cos(u)+v.*cos(u/2).*cos(u), ...
      sin(u)+v.*cos(u/2).*sin(u), ...
      v.*sin(u/2),'Edgecolor','blue');
view(15,56);
axis off;
hidden off;
```

### 3.3.4. Бутылка Клейна

```
[u,v]=meshgrid(linspace(0,pi,25),linspace(0,2*pi,50));
```



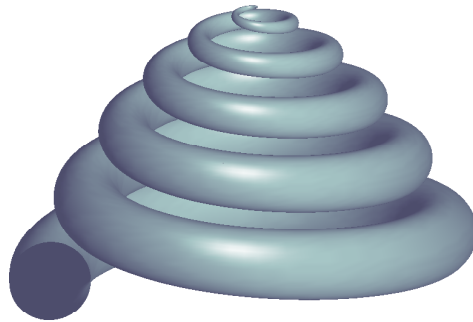


Рис. 3.20. Проволочная модель поверхности

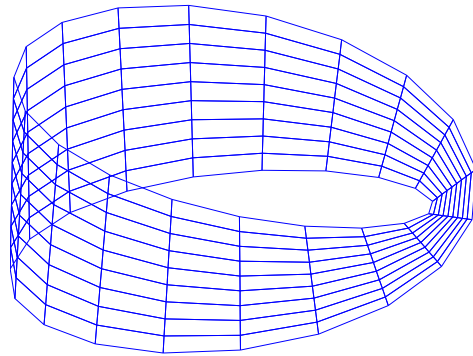


Рис. 3.21. Проволочная модель поверхности

```

r1 = 4*(1-cos(u)/2);
x1 = 6*cos(u).*(1+sin(u))+r1.*cos(u).*cos(v);
y1 = 16*sin(u)+r1.*sin(u).*cos(v);
z1 = r1.*sin(v);

[u,v]=meshgrid(linspace(pi,2*pi,25),linspace(0,2*pi,50));

r2 = 4*(1-cos(u)/2);
x2 = 6*cos(u).*(1+sin(u))+r2.*cos(v+pi);
y2 = 16*sin(u);
z2 = r2.*sin(v);

```

```

surfl(x1,y1,z1);
hold on;
surfl(x2,y2,z2);
view(9,21)
colormap pink;
shading interp;
hold off;
axis equal;
axis off;

```

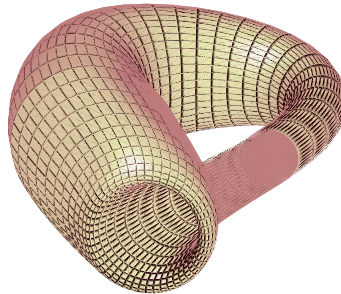


Рис. 3.22. Проволочная модель поверхности

### 3.4. Линии уровня

```

[X,Y]=meshgrid(-3:.25:3,-3:.25:3);
Z=sin(X).*sin(Y);
contour(X,Y,Z);

contourf(X,Y,Z);

```

### 3.5. Make it easy

```

ezplot('cos(tan(x))',-3,3)
ezplot('x^2+y^2-1')
ezplot('x-sin(y)')
ezplot('sin(t)/t','cos(t)/t',[1,10*pi])

```

```

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

f='sin(x^2+y^2)+exp(-x^2)';
d=[-2, 2, -2, 2];
ezmesh(f, d);
ezmeshc(f, d);
ezsurf(f, d);
ezsurfc(f, d);
ezcontour(f, d);
ezcontourc(f, d);

ezpolar('exp(cos(t))-2*cos(4*t)+sin(t/12)^5', [0, 6*pi])

```

### 3.6. Изображения (images)

Двумерные массивы могут быть отображены как образы, в этом случае элементы массива определяют яркость или цвет образов.

```

load durer
whos

```

Команды

```

image(X)
colormap(map)
axis image

```

изображают известную гравюру Дюрера. Элементы массива  $X$  — целые числа в диапазоне от 1 до 128 — рассматриваются как номера в палитре `map`. Команда

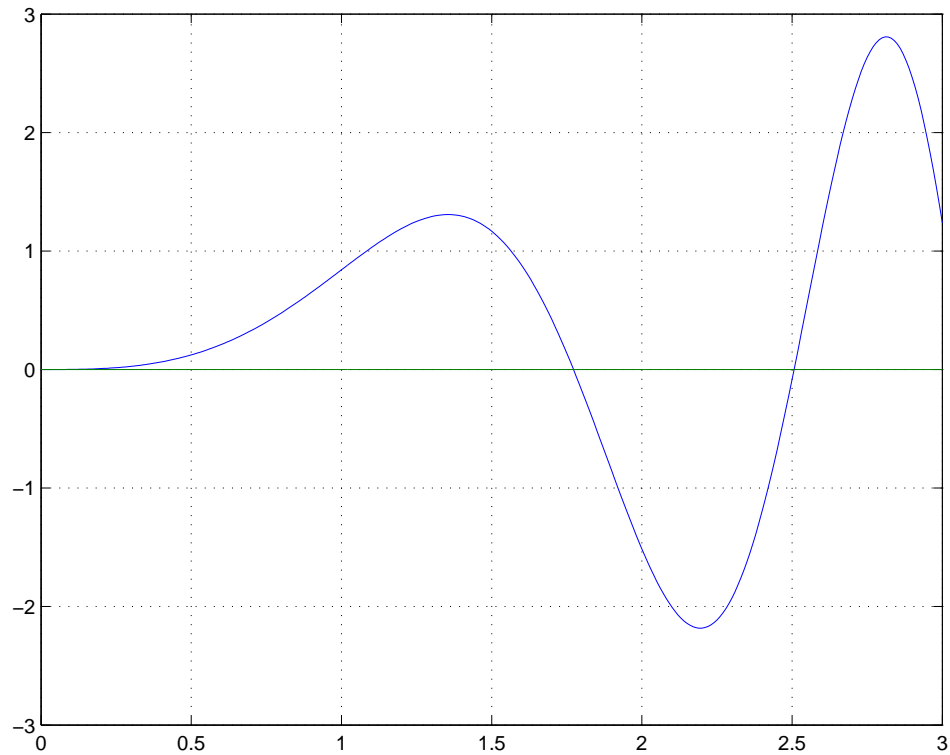
```
colormap(hot)
```

добавляет в гравюру современные цвета.

```
imagesc(rand(10000))
```

### 3.7. Графический способ решения уравнений

Функция `ginput` позволяет “снимать” координаты точек, указанных курсором мышки в графическом окне. По этой команде текущее графическое окно становится активным, курсор мышки принимает вид перекрестия. Координаты положения курсора в момент нажатия на левую

Рис. 3.23. Графический способ решения уравнения  $x \sin x^2 = 0$ 

кнопку мыши запоминаются. Выйти из этого режима можно нажатием на клавишу **Enter**.

С помощью команды `ginput` легко графически решать уравнения. Например, найдем корни уравнения

$$x \sin x^2 = 0$$

на отрезке  $[0, 3]$

```
x=0:.01:3; f=x.*sin(x.^2);
plot(x, [f; zeros(size(f))]),
grid; ginput
```

МАТЛАВ возвратит значения  $(0.0035, -0.0263)$ ,  $(1.7730, -0.0263)$ ,  $(2.4988, -0.0263)$ . Проверим найденные значения:

```
nx=length(x); w=1:nx-1;
x(find(f(w).*f(w+1)<0 | f(w)==0))
```

Для нахождения корней уравнения можно использовать функцию **fzero**. Команда

```
x=fzero('function',x0)
```

возвращает найденное значения корня функции **function** при заданном начальном приближении  $x_0$ .

Уточним найденные значения корня:

```
fzero('x.*sin(x.^2)', 1.8)
fzero('x.*sin(x.^2)', 2.5)
```

$$(z - z_1)(z - z_2) = 0, \quad z_1 = 0.5 + 0.3i, z_2 = 0.5 + 0.8i$$

```
x=0:.02:1; y=x;
[X,Y]=meshgrid(x,y);
Z=X+i*Y;
f=(Z-.5-.3*i).*(Z-.5-.8*i);
v=[0,0];
imagesc(abs(Z)); colorbar
contour(x,y,real(f),v)
hold on
contour(x,y,imag(f),v,'y')
% try sin(f), exp(f)
```

#### 3.7.1. Команда hold on

```
[x,y,z] = peaks;
contour(x,y,z,20,'k')
hold on
pcolor(x,y,z)
shading interp
hold off
```

### 3.8. Относительный и абсолютный способ задания цвета

Цвет различных графических объектов в системе MATLAB можно задавать несколькими разными способами:

- абсолютный способ задания цвета (true colors);
- относительный способ задания цвета (indexed colors):

- прямой,
- масштабированный.

В первом случае цвет задается как тройка чисел — RGB-вектор. Каждое из этих чисел заключено между нулем и единицей, оно характеризует долю красной, зеленой и синей составляющей в цвете соответственно.

Во втором случае фиксируется некоторая таблица  $k \times 3$  — палитра. Чтобы задать цвет нужно либо напрямую указать индекс — номер строки этой палитры (прямой относительный способ задания цвета), либо отмасштабировать имеющееся значение (масштабированный относительный способ). Поясним последний способ. Если имеется набор индексов (например, соответствующих цветам каждой грани объекта *Patch*), то выбирается такой линейный масштаб, что минимальный индекс указывает на первый элемент палитры, а максимальный — на последний элемент палитры.

### 3.9. Неявные функции

Покажем теперь на примере, как можно строить графики неявных функций. Рассмотрим кривую

$$f(x, y) = x^3y - 2xy^2 + y - 0.2 = 0, \quad x, y \in [0, 1].$$

```
h=.02; x=0:h:1; [X,Y]=meshgrid(x);
f=X.^3.*Y-2*X.*Y.^2+Y-.2;
v=[0,0]; contour(x,x,f,v), grid

v=[-.05,0,.05]; contour(x,x,f,v), grid
S=h^2*sum(f(:)>=0)

mesh(x,x,f)
g=f; g(f<0)=0; mesh(x,x,g)

C=contour(x,x,f); clabel(C)
```

## Глава 4

# Более сложные элементы языка MATLAB

### 4.1. Суммы

Функция `sum(a)` возвращает сумму элементов вектора `a`. Если `a` — это матрица, то функция возвращает сумму элементов в каждом столбце.

Функция `cumsum(a)` возвращает вектор той же длины, что и `a`, причем  $i$ -ая компонента возвращаемого вектора — это сумма первых  $i$  элементов вектора `a`.

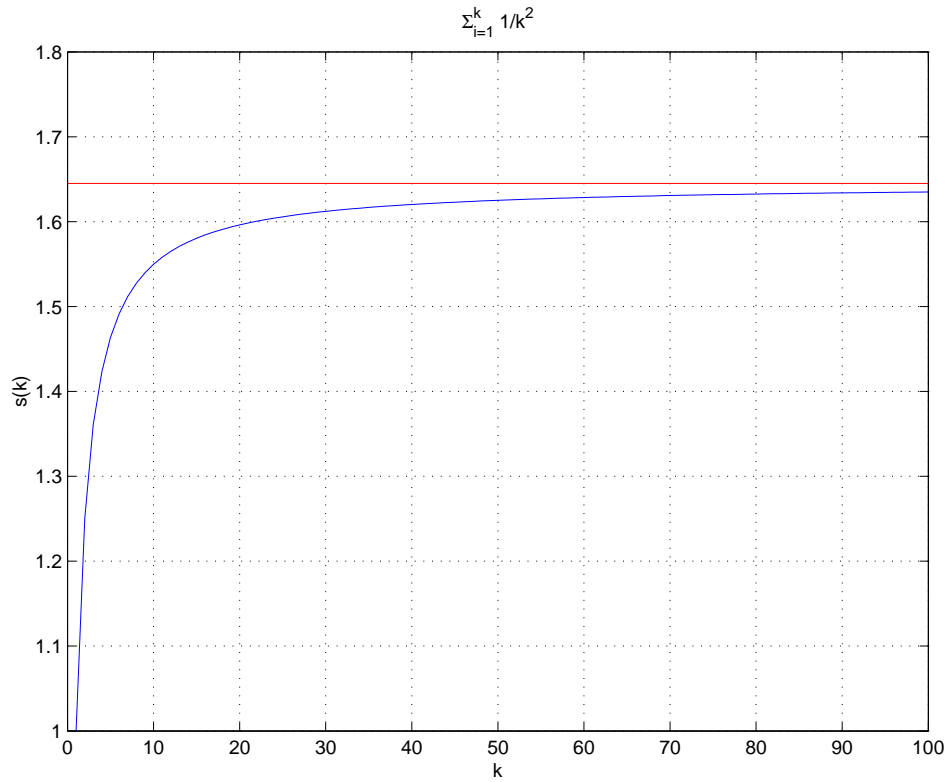
Рассмотрим пример:

```
a=1:20
b=cumsum(a)
plot(a, a, a, b)
```

Исследуем сходимость ряда:

$$\sum_{k=1}^{\infty} \frac{1}{k^2} = \frac{\pi^2}{6}$$

```
n=100; k=1:n; f=k.^(-2);
plot(cumsum(f));
l=pi^2/6; [sum(f), l]
hold on;
plot([0 n], [l l], 'r');
xlabel('k'); ylabel('s(k)');
title('\Sigma_{i=1}^k {1}/{k^2}');
% \sum не работает
```

Рис. 4.1. Сходимость ряда  $\sum_{k=1}^{\infty} \frac{1}{k^2}$  к  $\frac{\pi^2}{6}$ 

Рассмотрим более общий случай:

$$g(m) = \sum_{k=1}^{\infty} \frac{1}{k^m}$$

```
n=1000; k=1:n;
m=2; f=k.^(-m); plot(cumsum(f))
m=1.5; f=k.^(-m); plot(cumsum(f))
m=1.2; f=k.^(-m); plot(cumsum(f))
```

## 4.2. Произведения

Для нахождения произведения элементов массивов в системе MATLAB есть функции

```
prod(a)
cumprod(a)
```



аналогичные функциям `sum(a)`, `cumsum(a)`.

Рассмотрим бесконечное произведение

$$\prod_{k=2}^{\infty} \left(1 - \frac{1}{k^2}\right) = \frac{1}{2}$$

и экспериментально исследуем его скорость сходимости:

```
n=100;
k=2:n;
a=1-1./k.^(-2);
cp=cumprod(a);
cp(end),
plot(cp/.5) % относительная ошибка
```

### 4.3. Факториал

Чтобы найти факториал числа `m`, можно воспользоваться командой `\prod(1:m)`. Найдем, для какого максимального значения `m`, вычисление `m!` не даст переполнения:

```
n=200;
m=1:n;
mf=cumprod(m);
plot(mf, 'r')
sum(isinf(mf))
sum(isfinite(mf))
```

Функция `isinf` возвращает 1, если ее аргумент — это `inf`, и возвращает 0, в противном случае. Функция `isfinite` возвращает 1, если ее аргумент — это не `inf` и не `NaN`, и возвращает 0, в противном случае. Итак, максимальное значение `m`, для которого вычисление `m!` не дает переполнения — это 170.

### 4.4. Сочетание произведений и суммирования

Покажем, как не используя сложных команд управления логикой программы (циклы, ветвления, выбор), можно вычислять достаточно сложные выражения. Исследуем экспериментально скорость сходимости ряда

$$\sum_{k=1}^{\infty} \frac{x^k}{k!} = e^x, \quad x \in [-10, 30]. \quad (4.1)$$

Выполним следующие команды:

```

m=25; k=1:m; ki=1./k;
a=-10; b=30; h=.1; x=a:h:b;
M=[ones(size(x)); ki'*x];

```

Теперь матрица  $M$  имеет вид:

$$M = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ \frac{a}{1} & \frac{a+h}{1} & \frac{a+2h}{1} & \dots & \frac{b}{1} \\ \frac{a}{2} & \frac{a+h}{2} & \frac{a+2h}{2} & \dots & \frac{b}{2} \\ \vdots & \vdots & \vdots & & \vdots \\ \frac{a}{m} & \frac{a+h}{m} & \frac{a+2h}{m} & \dots & \frac{b}{m} \end{pmatrix}$$

Выполнив команду `cumprod(M)`, получим:

$$\text{cumprod}(M) = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ \frac{a}{1} & \frac{a+h}{1} & \frac{a+2h}{1} & \dots & \frac{b}{1} \\ \frac{a^2}{2!} & \frac{(a+h)^2}{2!} & \frac{(a+2h)^2}{2!} & \dots & \frac{b^2}{2!} \\ \vdots & \vdots & \vdots & & \vdots \\ \frac{a^m}{m!} & \frac{(a+h)^m}{m!} & \frac{(a+2h)^m}{m!} & \dots & \frac{b^m}{m!} \end{pmatrix}$$

В каждом столбце матрицы будут записаны члены ряда для разных  $x$ . Чтобы найти частичные суммы достаточно выполнить команду `cumsum`:

```

yt=exp(x); yv=sum(cumprod(M)); r=yv./yt;
plot(x, log(abs(r))), grid

```

Из графика видно, что сходимость неудовлетворительная при  $x < -7$  и  $x > 17$ . Найдём  $x$ , для которых  $e^x$  вычислена с точностью 3 верных знака:

```

q=1e-3;
v=-q<log10(abs(r)) & log10(abs(r))<q;
plot(r(v)), [min(x(v)), max(x(v))]

```

Причина краха: результат мал, а первые члены ряда очень большие:

```

x=-10; m=170; % try m=25,50, x=-20
k=1:m; M=[1;x./k'];
r=cumsum(cumprod(M))/exp(x);
plot(r), grid, r(end)-1

```

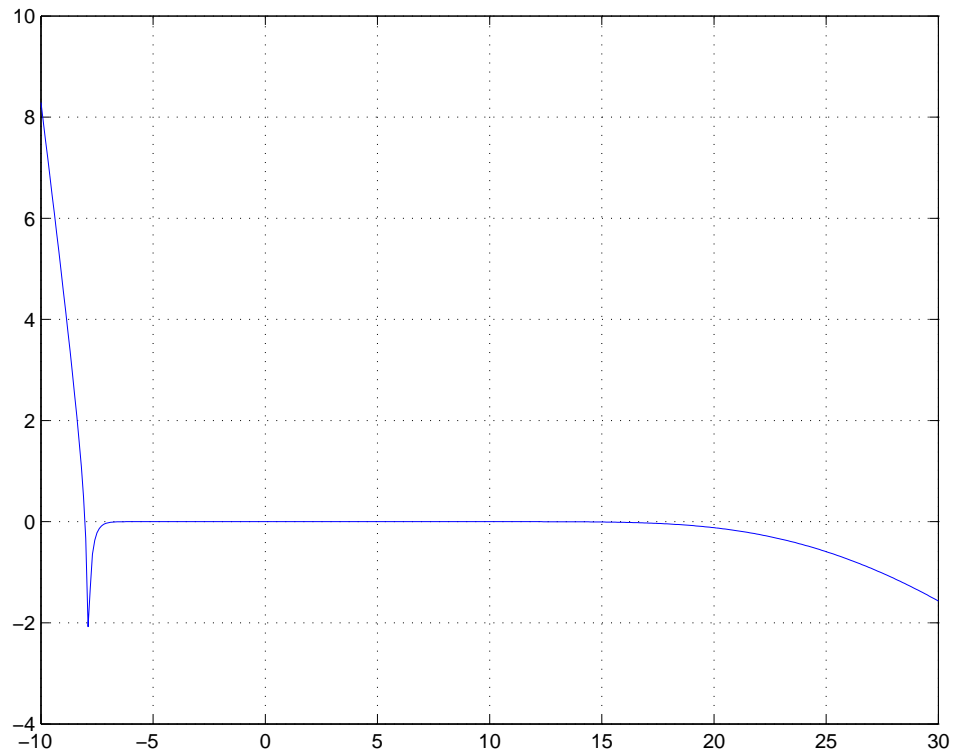


Рис. 4.2. График относительной ошибки для сходимости ряда (4.1) в логарифмическом масштабе

## 4.5. Интегрирование

С помощью команды `sum` можно методом прямоугольников численно вычислять квадратуры. Рассмотрим, например, интеграл

$$I = \int_0^3 \frac{x}{\sin x} dx$$

```
n=100;
h=3/n;
x=h/2:h:3;
f=x./sin(x);
plot(x,f);
plot(h*cumsum(f)),
grid;
sum(h*f)
```

В системе MATLAB есть встроенные функции для вычисления квадратур. Функция `trapz` использует метод трапеций:

```
hold on;
plot(cumtrapz(x, f), 'r'),
trapz(x, f)
```

Функция `quad` использует правило Симпсона с автоматическим выбором шага для вычисления квадратуры заданной точностью:

```
quad('x./sin(x)', eps, 3) % tol=1e-3
quad('x./sin(x)', eps, 3, 1e-4)
```

#### 4.5.1. Интегрирование функций с особенностью

$$I(a) = \int_0^1 x^a dx, \quad -1 < a < 0$$

#### 4.5.2. Аналитический учет особенности:

```
n=20; a=-.5; h=1/n; x=h:h:1; fx=x.^a;
plot(x,fx);
r=h^(a+1)/(a+1)+h*cumsum(fx);
[r(end) 1/(a+1) r(end)*(a+1)], plot(r)
% try n=200, a=-.99, n=20
```

#### 4.5.3. “Численный” учет особенности:

```
n=20; a=-.5; h=1/n; x=h/2:h:1; fx=x.^a;
r=h*cumsum(fx);
[r(end) 1/(a+1) r(end)*(a+1)], plot(r)
% try n=200, a=-.99, n=20
```

При  $a = -.99$  без аналитического учета особенности вычислить данный интеграл при разумном  $n$  невозможно.

```
n=20; a=-.5; h=1/n; x=h/2:h:1; fx=x.^a;
r=h*trapz(fx)
```

```
quad('x.^(-0.5)', 0, 1)
quad8('x.^(-0.5)', 0, 1)
quad('x.^(-.99)', 0, 1)
quad8('x.^(-.99)', 0, 1)
```

### 4.6. Управляющие конструкции

В данном разделе мы рассмотрим управляющие конструкции языка MATLAB, управляющие логикой вычислений. К ним относятся:

- условный оператор `if`,
- оператор цикла `while`,
- оператор цикла с параметром `for`,
- оператор выбора `switch`.

#### 4.6.1. Оператор `if`

```
if <условие1>
    <команды1>
elseif <условие2>
    <команды2>
elseif <условие3>
    <команды3>
...
else
    <команды4>
end
```

<условие> можно получить в результате логических операций `< >`, `==`, `<=`, `>=`, `~=`, `&`, `|`, `~` и специальных функций (`any`, `all`, ...)

Если `V` — массив, то условие `V` эквивалентно условию `all(V)`.

#### 4.6.2. Оператор `while`

```
while <выражение>
    <операторы>
end
```

Вычислим константу `eps`:

```
e=1;
while 1+e~=1,
    e=e/2;
end;
e=2*e
```

#### 4.6.3. Оператор for

```
for <переменная> = <выражение>
    <операторы>
end
```

Для примера вычислим матрицу Гильберта:

```
n=10;
H=zeros(n,n);
for i=1:n
    for j=1:n
        H(i,j)=1/(i+j-1);
    end;
end;
H % матрица Гильберта
```

#### 4.6.4. Оператор switch

```
switch <выражение>
case {<список значений 1>}
    <операторы>
case {<список значений 2>}
    <операторы>
...
otherwise
    <операторы>
end
```

Рассмотрим пример:

```
switch lower(method)
case {'linear','bilinear'}
    disp('Method is linear')
case 'cubic'
    disp('Method is cubic')
case 'nearest'
    disp('Method is nearest')
otherwise
    disp('Unknown method')
end
```

### 4.7. Хронометрирование

Познакомимся с некоторыми командами системы МАТЛАВ, позволяющими измерять время. Команда `toc` возвращает время в секундах, прошедшее после последнего вызова функции `tic`.

Например, измерим время, необходимое системе для вычисления матрицы Гильберта 1000-го порядка с помощью встроенной функции:

```
tic; h=hilb(1000); toc;
```

На компьютере автора на это ушло около 0.64 с.

Пока мы не умеем даже приближенно оценить, какое количество элементарных операций выполняет система для выполнения той или иной функции. Попробуем это сделать.

Для вычисления определителя квадратной матрицы порядка  $n$  система МАТЛАВ использует метод Гаусса, как известно, использующий

$$\sim \frac{2n^3}{3} \text{ арифметических операций.}$$

На основе этой информации попробуем оценить “среднее” время для выполнения 1 миллиона арифметических операций:

```
clear all;n=200;M=magic(n);
tic,det(M);t=toc/(2*n^3/3)*1e6
```

На компьютере автора получилось 0.0056 с.

Выбор  $n = 100$ , в действительности, был несколько случайным, но как мы увидим далее, правильным. Действительно, как показывает следующая программа, отношение

$$t(n) \bigg/ \frac{2n^3}{3}, \quad (4.2)$$

где  $t(n)$  — время вычисления определителя порядка  $n$ , не является константой.

```
clear all;
m=[50:25:1000];
M=1;
time=[];
for n=m;
    n,
    clear M;
```

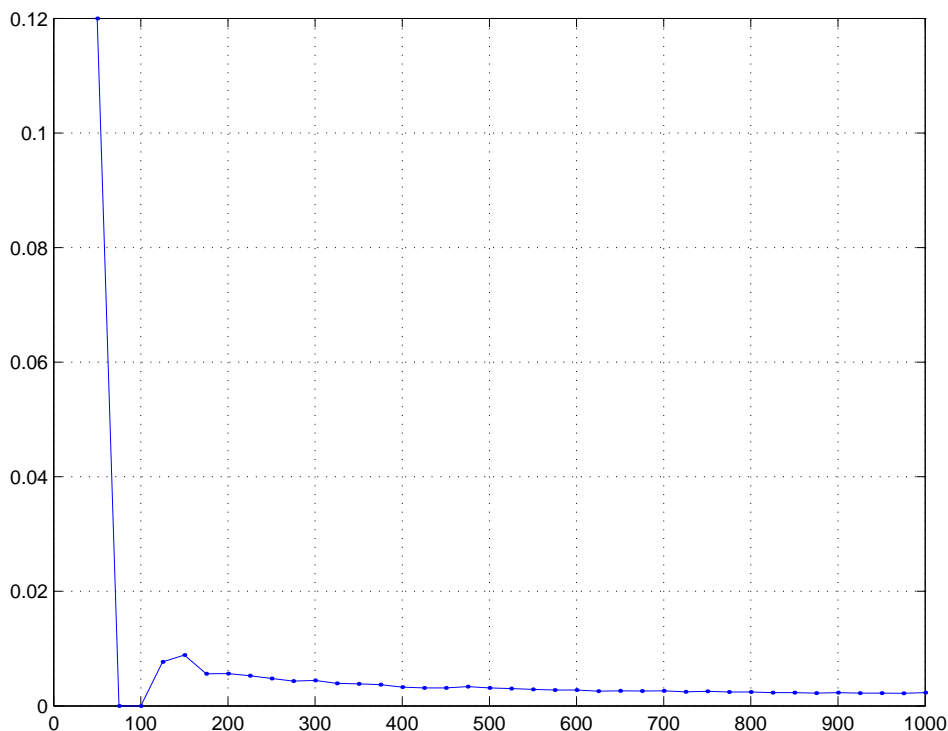


Рис. 4.3. Ускорение при вычислении определителя

```

M=magic(n);
tic; det(M); t=toc;
time=[time, t];
end

plot(m, time)
plot(m, time./(2.*m.^3/3)*1e6, '-.')
```

На графике мы видим скачки при небольших  $n$ . Начиная с  $n = 200$  отношение (4.2) монотонно убывает. Ускорение, которое мы получаем при росте  $n$ , объясняется использованием в системе MATLAB конвейерных вычислений, обеспечивающих форсирование векторных и матричных операций.

Случай  $n = 200$  возьмем за эталон, считая, что в среднем для выполнения 1 миллиона элементарных операций требуется 0.0056 с. Конечно, такой способ является далеко не точным для оценки “усредненной скорости работы компьютера”, однако годится для прикидочных расчетов.



#### 4.7.1. Работа вдоль строк и вдоль столбцов

В системе MATLAB матричные операции вдоль строк заметно медленнее матричных операций вдоль столбцов. Проверим это:

```
clear all; n=1000; M=magic(n);  
tic, sum(M); toc  
tic, sum(M,2); toc
```

Объясняется это тем, что матрицы хранятся по столбцам.

#### 4.7.2. Вычисление элементарных функций

Оценим количество элементарных операций, используемых в среднем для вычисления функций  $e^x$  и  $\sin x$ :

```
clear all; x=1:0.1:1e5;  
el_op=.0056;  
tic; exp(x); toc/el_op  
tic; sin(x); toc/el_op
```

Мы видим, что вычисление экспоненты требует в среднем 23 операции, а вычисление синуса — 82 операции.

#### 4.7.3. Циклы

Найдем время для организации в системе MATLAB пустого цикла:

```
clear all; el_op=.0056;  
tic; for k=1:1e6; end; toc/el_op  
tic; v=1:1e6; for k=v; end; toc/el_op
```

Получим, что на выполнение одной итерации цикла требуется около 70 и 82 операций соответственно. Это очень много, поэтому использование команд управления в системе MATLAB следует избегать.

#### 4.7.4. Присваивания

Найдем количество операций, используемых системой для организации присваивания. Вначале рассмотрим присваивание скалярных значений:

```
clear all; el_op=.0056;  
b=2;  
tic; for k=1:1e6; a=b; end; toc/el_op
```

Мы видим, что для одного присваивания в цикле необходимо около 390 операций. Теперь перейдем к присваиванию матричных значений:

```
clear all; el_op=.0056;
b=ones(1000);
tic; for k=1:1e6; a=b; end; toc/el_op
```

Время, необходимое для выполнения присваивания практически не изменилось: для одного присваивания в цикле матричного значения размером  $1000 \times 1000$  необходимо около 400 элементарных операций! Этот удивительный на первый взгляд факт снова объясняется использованием конвейерных вычислений: когда система встречает присваивание  $a=b$ , она лишь дает команду для выполнения этой операции и, не дожидаясь окончания, присваивания, переходит к следующей итерации цикла и т.д. Таким образом, присваивания выполняются как-бы “параллельно” и успевают завершиться к тому моменту, когда придет следующая команда выполнять присваивание.

Картина резко изменится, если к присваиванию мы добавим арифметическую операцию:

```
clear all; el_op=.0056;
b=ones(1000);
tic; for k=1:100; a=b+b; end; t=toc;
t/(el_op*1e-4)
```

Теперь команда  $a=b+k$  не успевает завершиться к началу следующей итерации и на одно присваивание требуется гораздо больше (около 12 млн.) элементарных операций.

#### 4.7.5. Двумерные и одномерные индексы

```
clear all; n=1000; M=ones(n); v=2:n-1;
tic, M(v,v)=0; toc
sum(M(:))
```

```
clear all; n=1000; M=ones(n);
v=(2:n-1)'; v1=ones(n-2,1);
V=v(:,v1)+n*v1*(v'-1);
tic, M(V)=0; toc
sum(M(:))
```

## 4.8. Типы данных

Помимо рассмотренных типов данных: массивов чисел и массивов символов, — в системе MATLAB поддерживаются новые абстрактные типы: многомерные массивы, массивы структур и массивы ячеек.

### 4.8.1. Многомерные массивы

Начиная с версии 5.0 в системе MATLAB появилась поддержка многомерных массивов. Один из самых простых способов создать такой массив — это использовать функции наподобие `ones` и `zeros`. Например, команда

```
ones(3,3,3)
```

создает трехмерный массив  $3 \times 3 \times 3$ , заполненный единицами, а команда

```
zeros(1,2,3,4,5)
```

создает пятимерный массив  $1 \times 2 \times 3 \times 4 \times 5$ , заполненный нулями.

Доступ к элементам массива по индексу такой же, как и для двумерных массивов, при этом доступны команды `[]`, `:`, `end` и т.п.

### 4.8.2. Массивы структур

*Структурой* мы будем называть абстрактный тип данных, представляющий собой коллекцию значений (*полей*) разных типов, доступ к которым осуществляется по имени (*имени поля*).

*Массив структур* — это коллекция структур, доступ к которым происходит по индексу. Массив структур может быть одномерным, двумерным или многомерным. Каждая структура в массиве должна иметь одинаковый набор полей. В системе MATLAB необязательно, чтобы типы элементов, которые хранятся в одноименных полях одного массива совпадали.

Создадим структуру `S` с двумя полями `name` и `age`. Присвоим им значения:

```
S.name = 'Isaac Newton';  
S.age = 38;
```

Сейчас `S` — это скалярная структура, или массив структур размеров  $1 \times 1$ :

name	age
'Isaac Newton'	38

Мы видим, что набор полей структуры может изменяться динамически. Также динамически могут меняться размеры массива структур:

```
S(2).name = 'Blaise Pascal' ;
S(2).age = 23;
```

Теперь **S** представляет собой вектор-строку из двух элементов:

№	name	age
1	'Isaac Newton'	38
2	'Blaise Pascal'	23

Возможен другой способ создания структуры:

```
S(3) = struct('name', 'Carl F. Gauss', 'age', 43) ;
```

Теперь массив **S** имеет размеры  $1 \times 3$ :

№	name	age
1	'Isaac Newton'	38
2	'Blaise Pascal'	23
3	'Carl F. Gauss'	43

Добавим еще одно поле:

```
S(3).profession='mathematician'
```

Теперь в качестве массива **S** будем иметь:

№	name	age	profession
1	'Isaac Newton'	38	[]
2	'Blaise Pascal'	23	[]
3	'Carl F. Gauss'	43	'mathematician'

#### 4.8.3. Массивы ячеек

*Ячейкой* (cell) мы назовем контейнер, который может содержать в себе произвольный из рассмотренных типов данных (т.е. это может быть массив чисел с плавающей запятой, массив символов, массив структур и др.) *Массив ячеек* — это коллекция ячеек, доступ к которым происходит по индексу. Таким образом, массив ячеек может объединять разнотипные данные. Массив может быть одномерным, двумерным или многомерным.

Доступ к ячейкам осуществляется указанием после имени массива индекса элемента в фигурных скобках. Такая ссылка на элемент может появиться в левой части присваивания. Например, программа

```
for n = 1:5
    M{n} = hadamard(2^n);
end
M{2}
```

создает массив, содержащий матрицы Адамара порядка 2, 4, 8, 16, 32.

Другой способ создать массив ячеек — это перечислить его элементы построчно, разделяя элементы в одной строке пробелами или запятыми, а сами строки — точкой с запятой или символом перехода на новую строку. Все элементы должны быть заключены в фигурные скобки. Например,

```
A = hadamard(4)
C = {A sum(A) prod(A) 'matrix A'}
```

Чтобы создать массив пустых ячеек достаточно воспользоваться функцией `struct` с указанием размеров массива. Например, команда

```
M = cell(5,1)
```

создает массив  $5 \times 1$  (столбец), содержащий пустые ячейки:

```
[]
[]
[]
[]
[]
```

## Глава 5

# Многочлены

### 5.1. Основные команды

В системе МАТЛАВ нет специального типа данных представляющих многочлен. Многочлен

$$f(x) = a_0x^n + a_1x^{n-1} + \dots + a_{n-1}x + a_n$$

представляется в виде вектора-строки  $f = (a_0, a_1, \dots, a_n)$ . Условие  $a_0 \neq 0$  в командах не проверяется.

*Произведение* многочленов  $f(x)$  и  $g(x)$ , где

$$g(x) = b_0x^m + b_1x^{m-1} + \dots + b_{m-1}x + b_m,$$

есть многочлен

$$h(x) = f(x)g(x) = c_0x^{n+m} + c_1x^{n+m-1} + \dots + b_{n+m},$$

в котором

$$c_k = \sum_{i+j=k} a_i b_j$$

суть компоненты свертки (convolution) векторов  $f$  и  $g = (b_0, b_1, \dots, b_m)$ . Для умножения многочленов в системе МАТЛАВ можно воспользоваться функцией нахождения свертки `conv(f,g)`

*Деление* многочленов

$$f(x) = g(x)p(x) + r(x)$$

осуществляется командой

`[p,r]=deconv(f,g)`

*Разложение дроби на простейшие*

$$\frac{f(x)}{g(x)} = k(x) + \sum_j \frac{\text{res}_j}{x - x_j}$$

осуществляется командой

```
[res,x,k]=residue(f,g)
```

где **res**, **x** — векторы-столбцы, а **k** — вектор-строка.

Для осуществления обратного перехода — по простейшим дробям восстановить исходную — нужно воспользоваться командой

```
[f,g]=residue(res,x,k)
```

Чтобы восстановить коэффициенты многочленов по его корням:

$$f(x) = (x - x_1) \cdot (x - x_2) \cdot \dots \cdot (x - x_n),$$

— воспользуйтесь командой **f=poly(x)**. Если **A** — квадратная матрица, то функция **poly(A)** возвращает коэффициенты характеристического многочлена этой матрицы.

Команда

```
y=polyval(f,x)
```

вычисляет значения многочлена  $f(x)$  в точках **x**. Если **X** — квадратная матрица, то команда

```
Y=polyval(f,X)
```

вычисляет значение  $Y = f(X)$  многочлена от матрицы.

Команда

```
g=polyder(f)
```

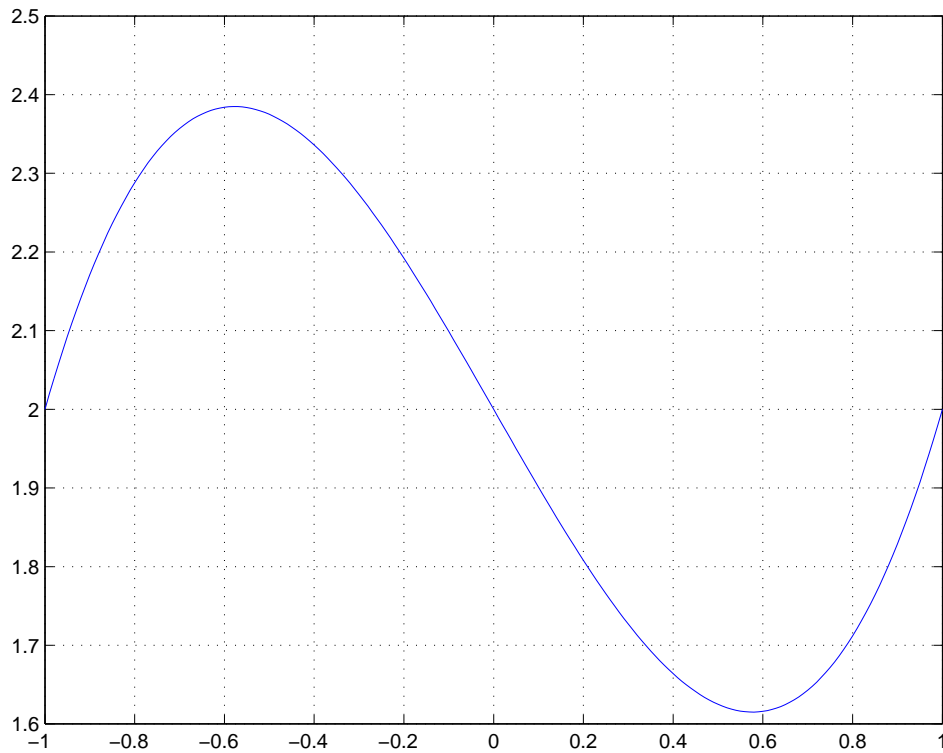
возвращает производную многочлена  $f(x)$ , а команда

```
g=polyder(f,k)
```

возвращает  $k$ -ую производную многочлена  $f(x)$ .

Для нахождения корней воспользуйтесь командой

```
roots(f)
```

Рис. 5.1. График многочлена  $f(x) = x^3 - x + 2$ 

## 5.2. Примеры

**Пример 1.** На примере многочлена

$$f(x) = x^3 - x + 2$$

покажем, как нарисовать его график и найти корни:

```
f=[1,0,-1,2];  
x=-1:.01:1;  
y=polyval(f,x);  
plot(x,y),  
grid  
roots(f)'
```

**Пример 2.** Разложим дробь

$$\frac{x^3 - x + 2}{x - 3}$$



на простейшие:

```
[g,r]=deconv(f,[1,-3])
```

Итак,

$$\frac{x^3 - x + 2}{x - 3} = x^2 + 3x + 8 + \frac{26}{x - 3}.$$

**Пример 3.** Разложим дробь

$$\frac{x - 3}{x^3 - x + 2}$$

на простейшие:

```
[res,x,k]=residue([1,-3],f);  
res',x',k
```

Получаем:

$$\frac{x - 3}{x^3 - x + 2} = -\frac{0.7607}{x + 1.5214} + \frac{0.3803 + 0.4289i}{x - 0.7607 - 0.8579i}$$

Восстановим по простейшим дробям исходную:

```
[q,p]= residue(res,x,k)
```

### 5.3. Сумма многочленов

В системе МАТЛАВ нет специальной команды для суммы многочленов, поэтому напомним свою функцию, выполняющую эту операцию:

```
function addpoly(f,g)  
  
nf=length(f);  
ng=length(g);  
n=max(nf,ng);  
s=[zeros(1,n-nf),f]+ [zeros(1,n-ng),g]
```

**Пример 4.** Проверим правильность функции на примере

$$f(x) = x^3 - x + 2, \quad g(x) = x + 3, \quad f(x) + g(x) = ?$$

```
f=[1,0,-1,2];  
g=[1,3];  
addpoly(f,g)
```

#### 5.4. Пример численной неустойчивости

**Пример 5.** Приведем классический пример численной неустойчивости при работе с многочленами (Уилкинсон, 1963 г.). Рассмотрим многочлен

$$f(x) = (x - 1) \cdot (x - 2) \cdot \dots \cdot (x - n)$$

с хорошо отделимыми корнями. Для разных  $n$  заставим МАТЛАВ явно вычислить коэффициенты этого многочлена, а затем найдем его корни:

```
n=2; x=(1:n)'; f=poly(x);
    xv=roots(f); R=[x,xv]
n=3; x=(1:n)'; f=poly(x);
    xv=roots(f); R=[x,sort(xv)]
max(abs(R(:,1)-R(:,2))./R(:,1))
plot(R), grid
```

При  $n = 2$  корни найдены точно, а при  $n = 3$  имеем 16 верных знаков. Возьмем  $n$  побольше,  $n = 10$ :

```
n=10; x=(1:n)'; f=poly(x);
    xv=roots(f); R=[x,sort(xv)]
format long, R
max(abs(R(:,1)-R(:,2))./R(:,1))
plot(R), grid
plot(R, '.'), grid
```

Точность ухудшилась, но по-прежнему остается великолепной: 10 верных знаков.

При  $n = 20$  — только 2 верных знака:

```
n=20; x=[1:n]'; f=poly(x);
    xv=roots(f); R=[x,sort(xv)]
max(abs(R(:,1)-R(:,2))./R(:,1))
plot(R),grid
```

Постараемся понять, почему снижается точность: объясняется ли это плохим алгоритмом, используемым в системе для нахождения корней многочлена, или это природа задачи? Для  $n = 20$  внесем малое возмущение  $10^{-7}$  в коэффициент  $f(2)$  при  $x^{19}$ . Такое возмущение (разумеется, не только в этот коэффициент) вполне могло возникнуть при начальном определении коэффициентов многочлена. Посмотрим, как возмущение скажется на корнях:

```
n=20; x=(1:n)'; f=poly(x);
f(2)=f(2)+1e-7; xv=roots(f); R=[x,sort(xv)]
max(abs(R(:,1)-R(:,2))./R(:,1)),
sum(abs(imag(xv))))
plot(R, '. '), grid
```

Крошечное изменение в исходном из коэффициентов приводит к появлению комплексных корней с относительно большой мнимой частью!

Попробуем объяснить, почему это произошло. Рассматриваемый многочлен имеет вид

$$f(x) = x^{20} + ax^{19} + \dots + 20!$$

Было внесено возмущение в коэффициент при  $x^{19}$ , т.е. в параметр  $a$ . Рассмотрим зависимость корней многочлена от этого параметра, т.е. рассмотрим функцию  $x(a)$ , неявно заданную уравнением  $f(x) = 0$ . По правилам взятия производной неявной функции,

$$\frac{dx}{da} = -\frac{\partial f / \partial a}{\partial f / \partial x}, \quad \text{где} \quad \frac{\partial f}{\partial a} = x^{19}$$

Частную производную  $\partial f / \partial x$  найдем с помощью функции `polyder(f)`:

```
n=20; x=[1:n]'; f=poly(x);
dfdx=polyder(f);
dxda=-(x.^19)./polyval(dfdx,x);
plot(dxda), grid
```

Мы получили график производной  $dx/da$ . В некоторых точках значение производной достигает громадной величины порядка  $10^9$ . Построим график производной в логарифмической шкале:

```
plot(log10(abs(dxda))),
grid
```

Итак, в нашем примере корни очень чувствительны к изменению коэффициентов многочлена. Так как при вычислении корней возникают ошибки округления, то мы можем считать, что решается не исходная, а некоторая близкая к исходной задача. Кроме того, уже исходные коэффициенты ввиду ошибок округления могли быть записаны неточно. Поэтому надеяться, что для чувствительной к начальным данным (или, как еще говорят, плохо обусловленной) задачи можно получить сверхточный ответ, бессмысленно.

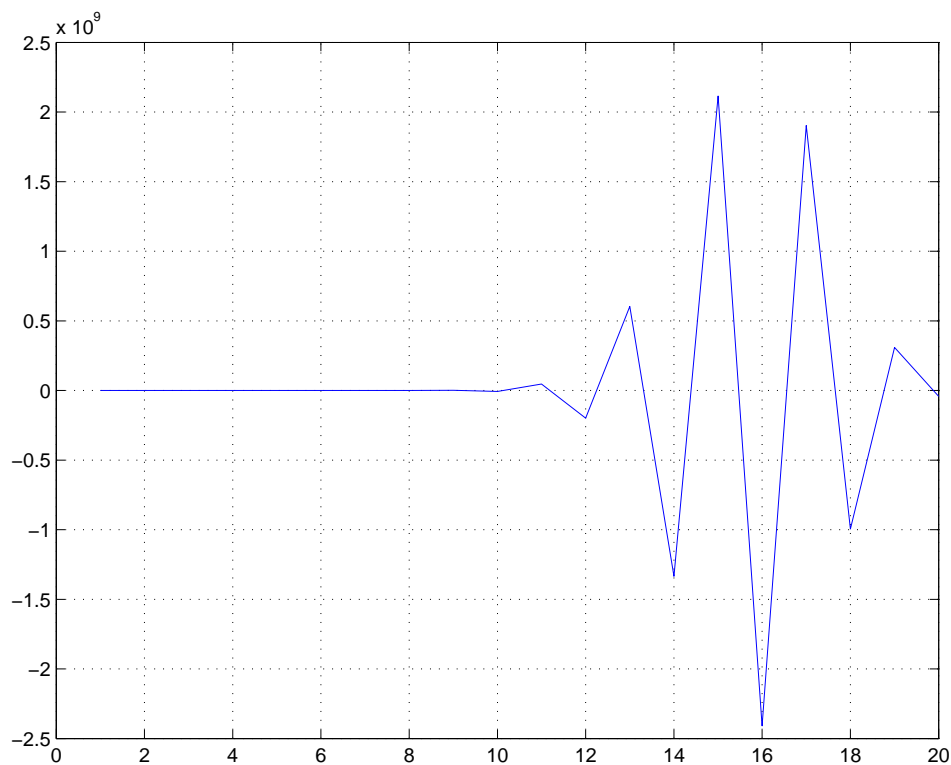


Рис. 5.2. График производной  $dx/da$ , выражающей степень зависимости корня многочлена Уилкинсона от параметра.

### 5.5. Многочлены Лежандра

$$L_0(x) = 1;$$

$$L_1(x) = x;$$

$$kL_k(x) = (2k-1)xP_{k-1}(x) - (k-1)P_{k-2}(x), \quad k > 1.$$

$$\|L_k(x)\| = \int_{-1}^1 L_k^2(x) dx = \frac{2}{2k+1},$$

$$(L_i, L_j) = \int_{-1}^1 L_i(x)L_j(x) dx = 0, \quad i \neq j$$

$$g(x) = \sum_{k=0}^m c_k L_k(x),$$

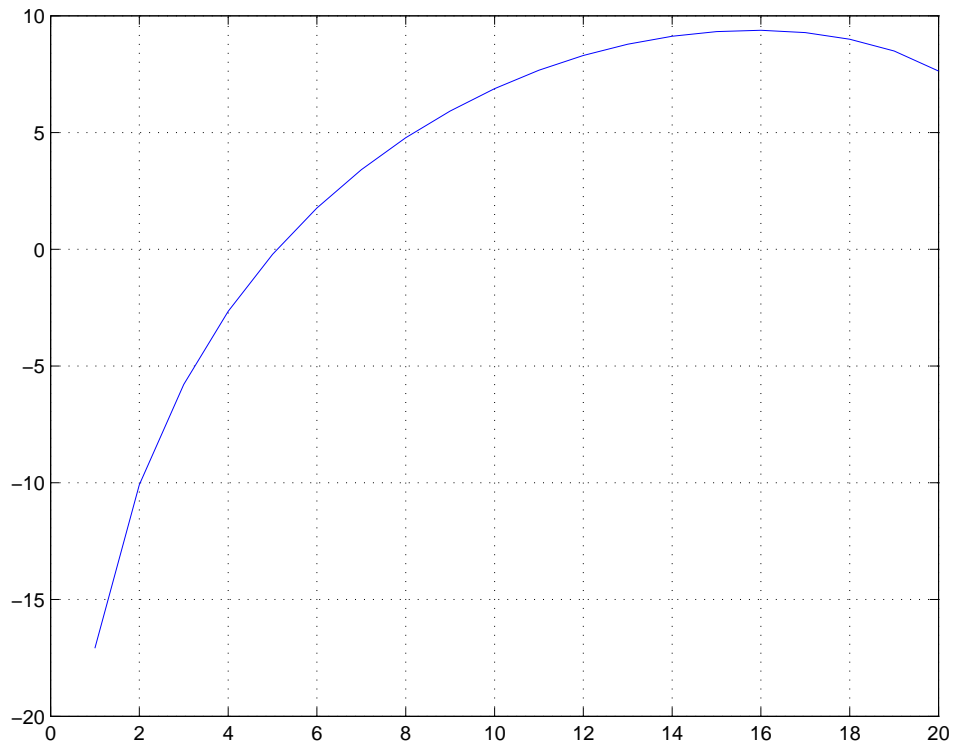
$$\text{где } c_k(x) = \frac{1}{\|L_k(x)\|} \int_{-1}^1 L_k(x)g(x) dx.$$

$$m=9;$$

$$n=100;$$

$$h=2/n;$$

$$x=-1+h/2:h:1-h/2;$$

Рис. 5.3. График функции  $\log_{10} |dx/da|$  в примере Уилкинсона.

```

P=[ones(size(x));x];
nKt=2./(2*(0:m)+1)';
K=3;
for k=1:m-1
    r=((2*k+1)*x.*P(K-1,:)-...
        k*P(K-2,:))/(k+1);
    P=[P;r];
    K=K+1;
end;
Q=h*P*P';

```

Графики

```

for K=1:m+1
    plot(x,P(K,:));
    grid;
    pause(2);

```

```
end;
plot(x,P);
```

Относительные ошибки

```
nK=diag(Q);
plot(nK./nKt);
```

Абсолютные ошибки

```
Q1=Q;
Q1(1:m+2:(m+1)^2)=0;
mor=max(abs(Q1(:)))
```

Приближение функции

```
g=7*x.^3-2*x+4;
c=(h*P*g') ./nKt;
gK=cumsum(c(:,ones(1,n)).*P);
M=[g;gK(end-5:end,:)];
plot(x,M), grid
```

Попробуем другие функции

$g(x) = x^2 - \sin(x)$	хорошее приближение
$g(x) =  x $	хорошее приближение
$g(x) = e^{2x}$	при $x < 0$ приближение хуже
$g(x) =  x ^{-0.5}$	плохое приближение

$$f(x) = (x - 2)(x_3) = x^2 - 5x + 6 = 0$$

$x = F(x)$ , где  $F(x) = (x^2 + 6)/5$ .

$$x_{k+1} = F(x_k)$$

```
xk=0;
n=100;
x=xk;
F='(xk.^2+6)/5';
for k=1:n;
    xk=eval(F)
    x=[x,xk];
end;
plot(x),
grid
```

$$F'(x) = \frac{2x}{5}, \quad F'(2) = 0.8 < 1, \quad F'(3) = 1.2 > 1$$

Оценим скорость сходимости

```
w=2:n;
v=(x(w+1)-x(w))./(x(w)-x(w-1));
plot(v);
grid
```

Сходимость для комплексных  $x_0$ ,  $|x_0| = 3$

```
clear i;
n=100;
xk=0;
x=xk;
phi=-pi:pi/20:pi;
xk=3*exp(i*phi)
F='(xk.^2+6)/5';
for k=1:n;
    xk=eval(F)
    x=[x,xk];
end;
plot(x, 'r.')
hold on;
plot(xk, 'r.', 'MarkerSize', 20)
grid
```

Граница области сходимости

```
r=3:.02:10;
phi=-pi:pi/90:pi;
z=[];
n=100;
for kr=r
    xk=kr*exp(i*phi);
    for k=1:n
        xk=eval(F);
    end
    z=[z;xk];
end;
zn=isnan(z);
```

```
X=r'*exp(i*phi);  
plot(X(zn)),  
axis equal;
```

Точный график

```
zn=isnan(z);  
zn=diff(zn)~=0;  
X=r'*exp(i*phi);  
plot(X(zn));  
hold on;
```

```
zn=isfinite(z);  
zn=diff(zn)~=0;  
X=r'*exp(i*phi);  
plot(X(zn), 'm');
```



## Матрицы и линейная алгебра

### 6.1. Простые команды

Напомним некоторые из команд построения матриц:

```
ones(n)    % матрица из единиц
zeros(n)    % матрица из нулей
eye(n)      % единичная матрица
rand(n)     % случайная матрица [0,1]
ones(m,n)
zeros(m,n)
eye(m,n)
rand(m,n)
```

В примерах будут использоваться команды построения теплицевой матрицы, т.е. матрицы вида

$$T(a, b) = \begin{pmatrix} a_1 & b_2 & b_3 & \dots & b_n \\ a_2 & a_1 & b_2 & \dots & b_{n-1} \\ a_3 & a_2 & a_1 & \dots & b_{n-2} \\ \dots & \dots & \dots & \dots & \dots \\ a_n & a_{n-1} & a_{n-2} & \dots & a_1 \end{pmatrix}.$$

Чтобы построить такую матрицу достаточно воспользоваться командой

```
toeplitz(a,b)
```

Команда

```
toeplitz(a)
```

эквивалентна

```
toeplitz(a,a)
```

**Пример 6.** Попробуйте выполнить команды:

```
toeplitz(1:4)
toeplitz(1:4,-1:-1:-5)
```

## 6.2. Решение систем линейных уравнений

Операции  $\backslash$  (левое деление) и  $/$  (правое деление) предназначены для решения систем линейных уравнений

$$Ax = b, \quad yA = b$$

соответственно. Чтобы найти решение достаточно выполнить команды:

```
x=A\b
y=b/A
```

Правая часть уравнений может быть не только вектором (столбцом для первой системы, строкой для второй системы), но и матрицей. В этом случае неизвестные — тоже матрицы и представляют собой наборы решений системы линейных уравнений с одинаковой левой частью и разными правыми частями.

**Пример 7.** Построим систему линейных уравнений с заранее известным ответом:

```
a=1:.1:5;
n=length(a);
A=toeplitz(exp(a));
xt=sin(a)';
b=A*xt;
```

Теперь найдем ее решение численно и нарисум точное решение и вычисленное:

```
x=A\b;
plot(xt);pause;plot(x)
```

Визуально точное и вычисленное решение совпали. Убедимся, что абсолютная и относительная ошибка ничтожна:

```

max(abs(xt-x))
max(abs(xt-x))/max(abs(xt))
norm(xt-x)
norm(xt-x)/norm(xt)

```

Для последнего выражения получим:  $4.4464 \cdot 10^{-13}$ . Найдем определитель системы:

```
det(A)
```

Он очень мал ( $4 \cdot 10^{-9}$ ).

Раньше было широко распространено мнение, что численно найти точное решение системы с определителем близким к нулю невозможно. Это заблуждение иногда встречается и сейчас, хотя установлено, что определитель играет только косвенную роль в этом вопросе (что и подтверждается предыдущим примером). Более полезной является другая характеристика системы (а точнее ее левой части) — *число обусловленности* матрицы  $A$ , которое является мерой чувствительности решения по отношению к исходным параметрам системы.

Чтобы дать определение числа обусловленности, напомним определение нормы вектора и нормы матрицы. Под *нормой вектора*  $x$  мы будем понимать величину

$$\|x\| = \sqrt{|x|^2},$$

а под *нормой матрицы*  $A$  — величину

$$\|A\| = \sup_{x \neq 0} \frac{\|Ax\|}{\|x\|}.$$

Из определения нормы матрицы следует, что

$$\|Ax\| \leq \|A\| \cdot \|x\|.$$

Можно показать, что

$$\|A\| = \max_{j=1,\dots,n} \lambda_j(A^T A) \leq \sqrt{\sum_{i,j=1}^n |a_{ij}|^2},$$

где через  $\lambda_j(B)$  ( $j = 1, \dots, n$ ) обозначены собственные числа матрицы  $B$ .

*Числом обусловленности* матрицы  $A$  называется величина

$$\text{cond } A = \|A^{-1}\| \cdot \|A\|.$$

Для симметричной матрицы  $A$  справедлива формула

$$\text{cond } A = \frac{\min |\lambda_j(A)|}{\max |\lambda_j(A)|}. \quad (6.1)$$

Пусть вместо системы  $Ax = b$  решается некоторая близкая “возмущенная” система  $(A + \Delta A)x = b + \Delta b$ . Пусть  $x$  — точное решение исходной системы, а  $x + \Delta x$  — точное решение “возмущенной” системы. Можно показать, что

$$\begin{aligned} \frac{\|\Delta x\|}{\|x\|} &\leq \text{cond } A \cdot \frac{\|\Delta b\|}{\|b\|} \quad \text{при } \Delta A = 0, \\ \frac{\|\Delta x\|}{\|x\|} &\leq \text{cond } A \cdot \frac{\|\Delta A\|}{\|A\|} \quad \text{при } \Delta b = 0. \end{aligned}$$

Таким образом, относительная ошибка  $\|\Delta x\|/\|x\|$  в решении системы линейных уравнений не больше некоторой величины, пропорциональной относительной ошибке в коэффициентах системы, с коэффициентом пропорциональности  $\text{cond } A$ .

Если даже коэффициенты системы были записаны абсолютно точно, то мы можем предположить, что ошибка округления накапливалась во время работы алгоритма, решающего систему линейных уравнений, и составила величину, равную машинному эпсилону  $\varepsilon_M$ , поэтому самое лучшее, что мы можем ожидать от алгоритма, это относительную ошибку величиной

$$\frac{\|\Delta x\|}{\|x\|} \approx \text{cond } A \cdot \varepsilon_M. \quad (6.2)$$

Например, при  $\varepsilon_M \approx 10^{-16}$  и  $\text{cond } A \approx 10^{14}$  получим относительную ошибку около  $10^{-2}$ , т.е. 2 правильные значащие цифры. Чем больше число обусловленности матрицы, тем более чувствительно решение системы по отношению к ее коэффициентам и, тем самым, менее точным будет ответ. Матрицы с большим числом обусловленности (например,  $\text{cond } A \geq \varepsilon_M^{-1/2}$ ) называются *плохо обусловленными*. Если матрица системы — плохо обусловлена, то сама система также называется плохо обусловленной.

Для вычисления числа обусловленности в системе MATLAB есть команда

`cond(A)`

Алгоритм вычисления числа обусловленности достаточно трудоемок (сложнее алгоритма Гаусса) и для оценки числа обусловленности часто используют более простой алгоритм, реализованный в функции `rcond`. Функция

```
r=rcond(A)
```

возвращает число  $r$ , такое, что  $\text{cond } A \leq 1/r$ , причем отношение  $1/r$  не бывает намного больше  $\text{cond } A$ .

**Пример 8.** Для системы из предыдущего примера функция

```
cond(A)
```

дает  $\text{cond } A = 6.6040 \cdot 10^3$ .

Так как матрица  $A$  симметрична, то для вычисления  $\text{cond } A$  можно воспользоваться формулой (6.1):

```
sp=eig(A);
y=min(abs(sp))
z=max(abs(sp))
c=z/y
```

Получим то же значение:  $\text{cond } A = 6.6040 \cdot 10^3$

Применим функцию `rcond`:

```
1/rcond(A)
```

Получим значение  $1.1294 \cdot 10^4$ .

Попробуем воспользоваться оценкой (6.2):

```
cond(A)*eps
```

Получим  $1.4664 \cdot 10^{-12}$ , что неплохо согласуется с относительной ошибкой  $4.4464 \cdot 10^{-13}$ , вычисленной в предыдущем примере.

```
%спектр и число обусловленности (2-норма)
ssp=sort(sp);v=1:n-1;di=diff(ssp);
min(abs([di./ssp(v);di./ssp(v+1)]))
%0.0044
[Q,D]=eig(A);
L=Q'*Q;
L(1:n+1:n^2)=0;
max(abs(L(:)))
%проверка Q'*Q=E
[Q,D]=eig(A);
L=Q\A*Q-D;
max(abs(L(:)))
%проверка Q\A*Q=D
```

### 6.3. Спектр. Число обусловленности

```
d=eig(A)
[Q,D]=eig(A)
cond(A)
```

### 6.4. Нормы векторов и матриц

Число обусловленности  $Ax=b$   
 похожая формула справедлива и для относительной ошибки левой части

### 6.5. Матрицы Гильберта

Матрица  $H_n = (h_{ij})$  размеров  $n \times n$  с элементами

$$h_{ij} = \frac{1}{i+j-1} \quad (i = 1, \dots, n; j = 1, \dots, n)$$

называется *матрицей Гильберта*. Известно, что (для достаточно больших  $n$ ) матрицы Гильберта — плохо обусловлены. Матрица Гильберта является частным случаем матрицы Коши, которая определяется следующим образом: если  $a, b$  — два вектора длиной  $m$  и  $n$  соответственно, то *матрицей Коши* называется матрица  $(a, b) = (c_{ij})$  размеров  $m \times n$ , где

$$c_{ij} = \frac{1}{a_i + b_j} \quad (i = 1, \dots, m; j = 1, \dots, n).$$

Для построения матрицы Коши можно воспользоваться командой

```
C=gallery('cauchy',a,b)
```

**Пример 9.** [Матрицы Гильберта] Построим матрицы Гильберта малых порядков:

```
format rat;
for n=2:5
    gallery('cauchy',0:n-1,1:n)
end;
```

Например, при  $n = 4$  получим:

$$H_4 = \begin{pmatrix} 1 & 1/2 & 1/3 & 1/4 \\ 1/2 & 1/3 & 1/4 & 1/5 \\ 1/3 & 1/4 & 1/5 & 1/6 \\ 1/4 & 1/5 & 1/6 & 1/7 \end{pmatrix}$$

Исследуем рост числа обусловленности матриц Гильберта:

```
format;
c=[];
for n=2:20
    A=gallery('cauchy',0:n-1,1:n);
    c=[c;cond(A)];
end;
plot(c);
c
```

Мы видим, что число обусловленности принимает огромные значения: при  $n = 10$ , например, достигает величины порядка  $10^{15}$ . Таким образом, решая численно систему линейных уравнений с  $H_{10}$  мы не можем рассчитывать на точность в ответе большую, чем 1 значащая цифра.

Посмотрим какую оценку для числа обусловленности даст `rcond`:

```
r=[];
for n=2:20
    A=gallery('cauchy',0:n-1,1:n);
    r=[r;1/rcond(A)];
end;
plot(r);
r
```

Чтобы визуально сравнить результаты работы функций `cond` и `rcond`, построим графики  $\log_{10} c$  и  $\log_{10} r$ :

```
plot(log10([c r]));
grid;
```

Мы видим, что обе функции дают согласованные ответы.

Теперь для  $n = 5$  построим систему линейных уравнений с точным решением  $x_j = e^j$  ( $j = 1, \dots, n$ ) и найдем ее решение численно:

```
n=5;
A=gallery('cauchy',0:n-1,1:n);
xt=exp(1:n)';
b=A*xt;
x=A\b;
max(abs(x-xt))
e=norm(x-xt)
```

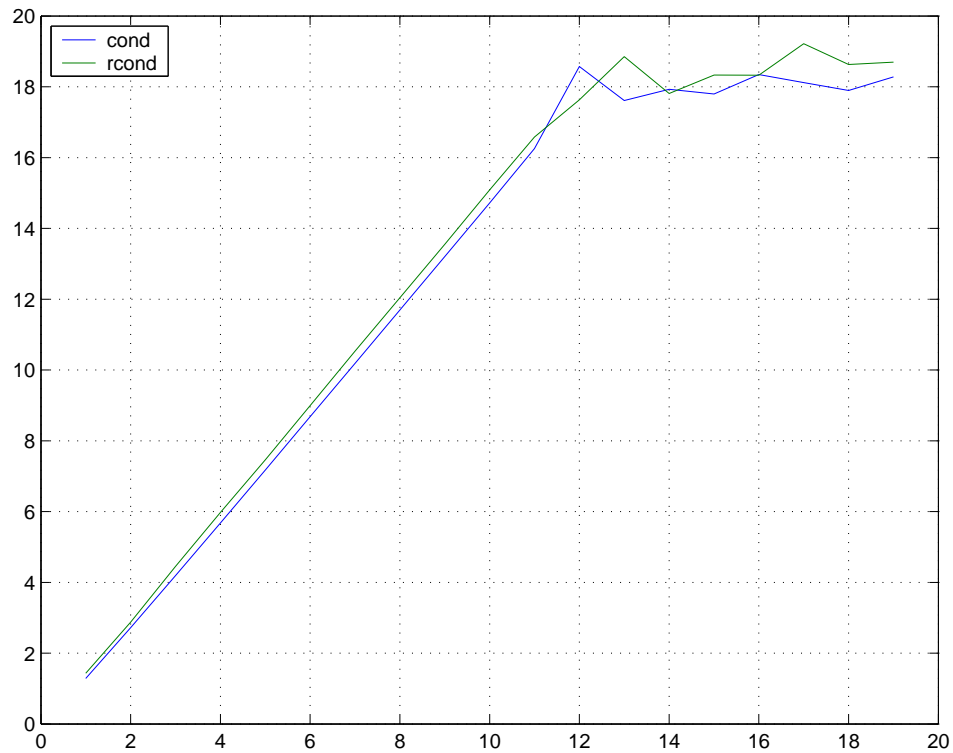


Рис. 6.1. Число обусловленности для матриц Гильберта в логарифмической шкале. Для вычисления использовались функции `cond`, `rcond`

Получим погрешность  $e \approx 10^{-9}$ . Это согласуется с формулой (6.2), так как

```
cond(A)*eps
```

дает величину порядка  $10^{-10}$ .

## 6.6. Экспериментальное исследование числа обусловленности

Сконструируем систему линейных уравнений  $Ax = b$  с известным точным решением  $x_t$ :

```
h=.1;
a=1:h:5;
n=length(a);
A=toeplitz(exp(a));
```



```
xt=sin(a)';
b=A*xt;
x=A\b;
```

пусть ошибка — равномерная случайная величина:

```
me=1e-4*h*abs(cumsum(b));
rand('state',0);
m=30;
V=(1:n)'*ones(1,m);
plot([b/max(abs(b)),me/max(me)]);
grid;
```

m=30 возмущений правой части:

```
B=b(V)+me(V).*(2*rand(n,m)-1);
X=A\B;
plot(X)
plot([me(V).*(2*rand(n,m)-1),me(V)])
```

```
%
3;4;xu=max(X,[],2);xd=max(X,[],2);plot([xu,xt,xd])
%работаем вдоль строк
5;plot(B); zoom on %возмущений в правых частях не видно
6;sp=eig(A);c=max(abs(sp))/min(abs(sp))
cond(A) %0.6064, тогда как
7;max([xu-xt;xt-xd]) %0.5180
8;mme=max(me);plot(A\[b-mme,b,b+mme]) %слишком упрощенно
%линии пересекаются около i=2
8;mme=100*max(me);plot(A\[b-mme,b,b+mme])
```

Увеличение размеров матрицы

```
1;clear all;h=.01;a=1:h:5;A=toeplitz(exp(a));xt=sin(a)';b=A*xt;x=A\b;
plot([xt,x]),cond(A)
2;max(abs(x-xt)) %1.3153e-9
3;[Q,D]=eig(A);L=Q'*Q;n=length(a);L(1:m+1:m^2)=0;max(abs(D(:)))
```

# Интерполяция и аппроксимация данных

## 7.1. Интерполяция

Пусть про некоторую функцию  $g(x)$  известно, что она в точках  $x_1, x_2, \dots, x_n$  ( $x_1 < x_2 < \dots < x_n$ ) принимает значения  $y_1, y_2, \dots, y_n$ . Задача интерполяции заключается в построении такой функции  $p(x)$  из известного класса, которая в точках  $x_i$  принимает те же значения  $y_i$ , что и функция  $g(x)$ . Функцию  $p(x)$  часто называют интерполянтom, точки  $x_i$  — узлами интерполяции, величины  $y_i$  — значениями интерполяции. Выделим тот случай, когда  $p(x)$  — многочлен. Он называется интерполяционным. Известная теорема из теории аппроксимации говорит о том, что для произвольной таблицы интерполяции

$x_1$	$x_2$	$\dots$	$x_n$
$y_1$	$y_2$	$\dots$	$y_n$

интерполяционный многочлен  $p(x)$  степени меньше  $n$  существует и единственен. Если интерполируемая функция  $g(x)$  в некоторой области имеет  $n$  непрерывных производных то для любой точки  $x$  из этой области погрешность интерполяции равна

$$g(x) - p(x) = \frac{g^{(n)}(\xi)}{n!} (x - x_1)(x - x_2) \dots (x - x_n),$$

где  $\xi$  — некоторая точка, удовлетворяющая неравенству  $x_1 < \xi < x_n$ . Иногда данная формула позволяет оценить ошибку интерполяции.

В системе MATLAB коэффициенты интерполирующего многочлена можно найти с помощью функции `polyfit`. Если  $\mathbf{x}$  — массив, содер-

жащий  $n$  узлов, а  $y$  — массив, содержащий  $n$  значений, то коэффициенты интерполяционного многочлена можно найти с помощью команды `p=polyfit(x,y,n-1)`. К функции `polyfit` мы вернемся, когда будем изучать метод наименьших квадратов.

**Пример 10.** Функцию  $y = \ln x$  будем интерполировать кубическим полиномом по точкам 0.4, 0.5, 0.7, 0.8. Оценим погрешность интерполяции в точке 0.6. Для нахождения интерполианта выполним программу

```
x=[0.4 0.5 0.7 0.8];
y=log(x);
p=polyfit(x,y,3)
```

Получим:

```
p =
    1.6836    -4.5239     5.2760    -2.4106
```

Таким образом, интерполиант имеет вид:

$$p(x) = 1.6836x^3 - 4.5239x^2 + 5.2760x - 2.4106.$$

Выражение для погрешности дает:

$$\ln(0.6) - p(0.6) = -\frac{6}{\xi^4} \frac{1}{4!} (0.6 - 0.4)(0.6 - 0.5)(0.6 - 0.7)(0.6 - 0.8),$$

$$0.4 < \xi < 0.8.$$

Следовательно, ошибка для погрешности не больше

$$\frac{6}{0.4^4} \frac{1}{24} 0.0004 \approx 0.0039.$$

Найдем фактическую абсолютную ошибку:

```
polyval(p,0.6)-log(0.6)
```

Получим, что  $|p(0.6) - \ln 0.6| = 0.00085$ . В некоторых задачах не удастся найти даже границы ошибки интерполяции.

Попробуем экспериментально исследовать, что будет, если для интерполяции использовать все большее и большее число точек. Выражение для погрешности состоит из трех разных частей: факториала в знаменателе и производной с произведением разностей в числителе. Существуют примеры, когда производная с ростом степени  $n$  интерполирующего полинома растет быстрее  $n!$ . В этом случае использовать интерполиант высокой степени нельзя.

**Пример 11.** [Функция Рунге] Рассмотрим функцию

$$R(x) = \frac{1}{1 + 25x^2}$$

и попробуем на отрезке  $[-1, 1]$  найти интерполянт к ней по  $n = 1, 2, \dots, 12$  равномерно распределенным точкам:

```
xx=-1:.01:1;
yy=1./(1+25*xx.^2);
plot(xx,yy,'r','LineWidth',3),grid;

for n=1:12,
    x=linspace(-1,1,n);
    y=1./(1+25*x.^2);
    p=polyfit(x,y,n-1);
    yp=polyval(p,xx);
    hold on;
    plot(xx,yp,'k');
end;
```

С ростом  $n$  удовлетворительные результаты получаются лишь для  $x \in [-0.73, 0.73]$ . Вне этого отрезка интерполяционный процесс расходится. Объяснить это можно, например, тем, что производные  $R^{(n)}(\xi)$ , фигурирующие в оценке погрешности, быстро растут с ростом  $n$ :

```
df=diff(yy)/0.01;
max(df)
df2=diff(yy,2)/0.0001;
max(df2)
df3=diff(yy,3)/0.01^3;
max(df3)
df12=diff(yy,12)/(0.01)^12;
mdf12=max(df12)
```

Мы получили  $\max R'(x) \approx 3, 2$ ,  $\max R''(x) \approx 12, 5$ ,  $\max R'''(x) \approx 580$  и уже  $\max R^{(12)}(x) \approx 10^{17}$ . Формула для оценки погрешности интерполяции функции полиномом 11 степени дает погрешность  $8 \cdot 10^{11}$ . Это, конечно, *очень* завышенная оценка: реальная погрешность составляет 0.55:

```
mdf12*2^12/factorial(12)
max(abs(yp-yy))
```

Попробуем теперь сгустить точки около концов отрезка  $[-1, 1]$ . В качестве узлов интерполяции возьмем корни полиномов Чебышева:

$$x = \cos \frac{(2i-1)\pi}{2n} \quad (i = 1, 2, \dots, n).$$

```
hold off;
xx=-1:.01:1;
yy=1./(1+25*xx.^2);
plot(xx,yy,'r','LineWidth',3),grid;
hold on;

for n=15:20;
    i=1:n;
    x=cos((2*i-1)*pi/(2*n));
    sort(x);
    y=1./(1+25*x.^2);
    p=polyfit(x,y,n-1);
    yp=polyval(p,xx);
    plot(xx,yp,'k');
end;

plot(x,zeros(size(x)),'o');

hold off;
```

Затруднение с функцией Рунге исчезло. Абсолютная ошибка интерполяции этой функции полиномом 19 степени составляет 0.038, а относительная — лишь 0.004:

```
max(abs(yp-yy))
max(yp/yy)
```

Предыдущий пример показал, что использование равномерных узлов для интерполантов высокой степени может привести к неудовлетворительному результату. Теорема Фабера говорит, что не существует общего правила для выбора узлов, которое гарантировало бы сходимость  $p_n(x) \rightarrow g(x)$  при  $n \rightarrow \infty$  для любой непрерывной функции  $g(x)$ . В то же время по теореме Марцинкевича для каждой непрерывной функции можно индивидуально подобрать такие точки.

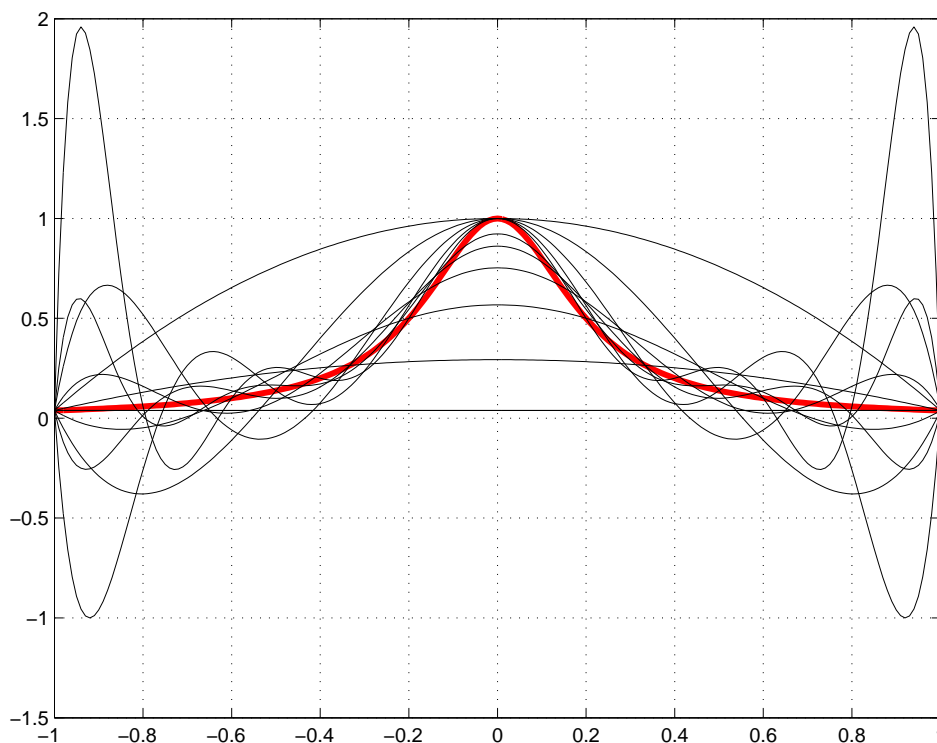


Рис. 7.1. Полиномиальная интерполяция функции Рунге с равномерным распределением узлов

## 7.2. Кусочно-полиномиальные функции

Функция  $f(x)$  называется кусочно-полиномиальной, если на каждом из отрезков  $[x_i, x_{i+1}]$  ( $i = 1, 2, \dots, n-1$ ;  $x_1 < x_2 < \dots < x_n$ ) она представляет собой многочлен

$$f_i(x) = a_{i1}x^k + a_{i2}x^{k-1} + a_{ik}x + a_{i,k+1}.$$

Часто удобно иметь дело не с многочленами  $f_i(x)$ , а с многочленами

$$h_i(x) = b_{i1}x^k + b_{i2}x^{k-1} + b_{ik}x + b_{i,k+1},$$

полученными из  $f_i(x)$  сдвигом отрезка  $[x_i, x_{i+1}]$  в новое положение  $[0, x_{i+1} - x_i]$ . Связь коэффициентов этих двух систем многочленов устанавливается очевидным равенством

$$h_i(x) = f_i(x + x_i).$$

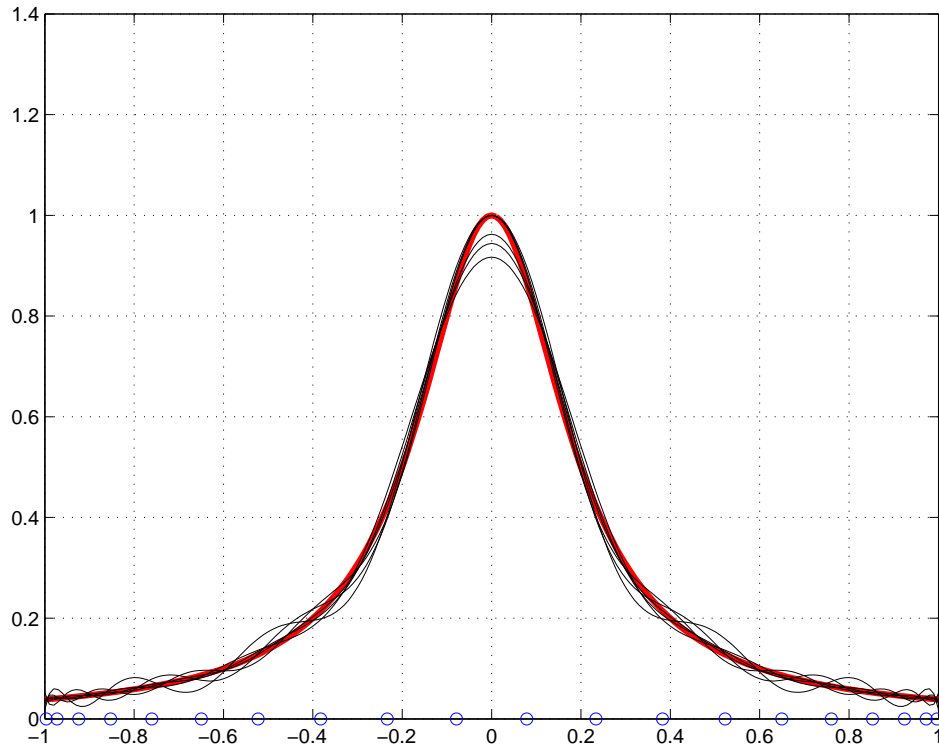


Рис. 7.2. Полиномиальная интерполяция функции Рунге с чебышевским распределением узлов

Для представления кусочно-полиномиальных функций система МАТЛАВ использует структуру специального вида. Эту структуру мы будем называть *pp*-представлением.

Функция `pp=mkpp(x,B)` создает *pp*-представление кусочно-полиномиальной функции по узлам  $x$  и коэффициентам многочленов  $B$ . Вектор  $x$  должен иметь длину  $n$  и содержать координаты узлов  $x_1, x_2, \dots, x_n$ . Матрица  $B$  должна иметь размеры  $n \times (k+1)$  и содержать коэффициенты многочленов  $h_i(x)$ :

$$B = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1,k+1} \\ b_{21} & b_{22} & \dots & b_{2,k+1} \\ \dots & \dots & \dots & \dots \\ b_{n-1,1} & b_{n-1,2} & \dots & b_{n-1,k+1} \end{pmatrix}$$

Функция `[x,B]=unmkpp(pp)` возвращает вектор узлов  $x$  и матрицу коэффициентов  $B$  по представлению *pp* кусочно-полиномиальной функции.

Функция `yy=ppval(pp,xx)` возвращает набор значений `yy` в точках `xx` кусочно-полиномиальной функции, заданной представлением `pp`.

**Пример 12.** Рассмотрим кусочно-квадратичную функцию

$$f(x) = \begin{cases} -x^2 + 4x, & \text{если } 0 \leq x \leq 2, \\ 4, & \text{если } 2 \leq x \leq 4, \\ -x^2 + 8x - 16, & \text{если } 4 \leq x \leq 6. \end{cases}$$

После сдвига получаем

$$\begin{aligned} h_1(x) &= -x^2 + 4x, \\ h_2(x) &= 4, \\ h_3(x) &= -(x+4)^2 + 8(x+4) - 16 = -x^2 + 4. \end{aligned}$$

Нарисуем графики функций  $h_i(x)$  и  $f(x)$ :

```
xx=[0:0.05:2];
yy1=polyval([-1 4 0],xx);
yy2=polyval([0 0 4],xx);
yy3=polyval([-1 0 4],xx);
subplot(2,3,2);
plot(xx,yy1,xx,yy2,xx,yy3);
axis([0 2 0 5]);
title('h_1(x), h_2(x), h_3(x)');

x=[0 2 4 6];
A=[-1 4 0;
    0 0 4;
    -1 0 4];
pp=mkpp(x,A);
xx=[0:0.05:6];
yy=ppval(pp,xx);
subplot(2,1,2);
plot(xx,yy);
axis([0 6 0 5]);
title('f(x)');
```

### 7.3. Кусочно-полиномиальная интерполяция

В этом разделе мы дадим краткий обзор некоторых возможностей системы МАТЛАВ по интерполяции данных с помощью кусочно-полиномиальных



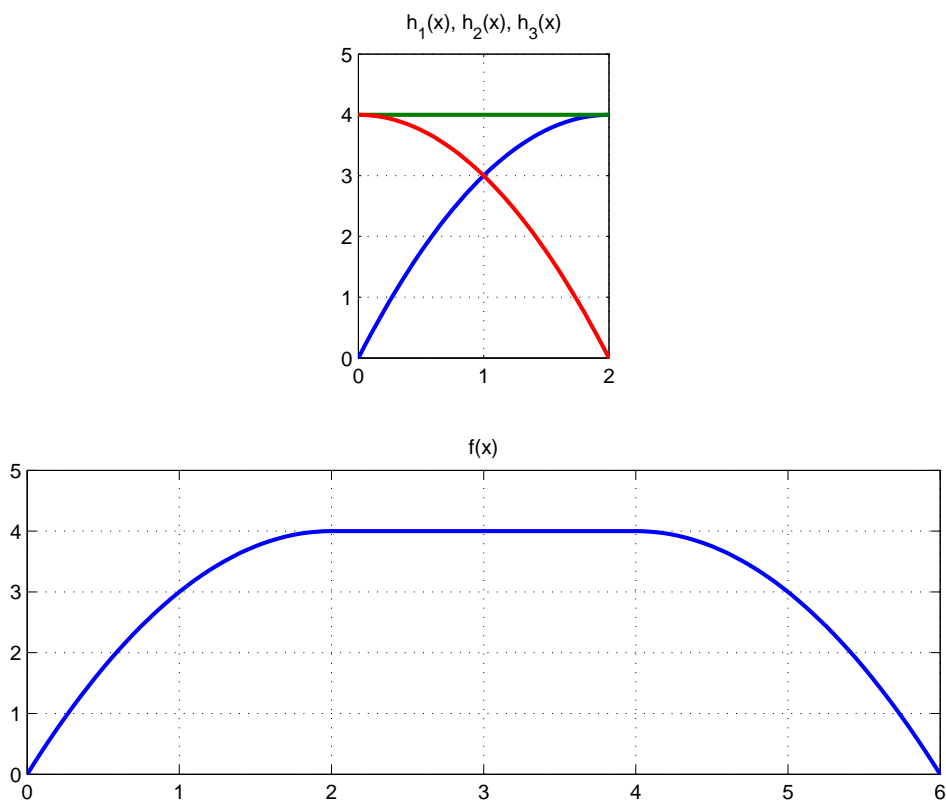


Рис. 7.3. Кусочно-полиномиальная функция

функций. В последующих разделах мы обсудим эти возможности более обстоятельно.

Следующие функции по узлам  $x$  и значениям в узлах  $y$  находят кусочно-полиномиальный интерполянт и возвращают его значения в точках  $xx$ :

- `interp1(x,y,xx,'nearest')` использует ступенчатую интерполяцию,
- `interp1(x,y,xx)` или `interp1(x,y,xx,'linear')` строит кусочно-линейный интерполянт,
- `spline(x,y,xx)` или `interp1(x,y,xx,'spline')` находит кубический сплайн,
- `pchip(x,y,xx)` или `interp1(x,y,xx,'pchip')` или `interp1(x,y,xx,'cubic')` ищет эрмитов кубический интерполянт,

- `interp1(x,y,xx,'v5cubic')` реализует алгоритм построения кубического интерполянта из системы МАТЛАВ 5.

При равномерной сетке  $x$  можно использовать более быстрые методы ‘со звездочкой’: `nearest*`, `linear*`, `spline*`, `pchip*` — например:

```
interp1(x,y,xx,'spline*')
```

Кроме этого, функции `pp=spline(x,y)` `pp=pchip(x,y)` возвращают `pp`-представление для сплайн-интерполянта и эрмитова кубического интерполянта соответственно.

**Пример 13.** Построим интерполянты функции Рунге по 9 точкам и вычислим абсолютные ошибки интерполяции:

```
x=-1:0.25:1;
y=1./(1+25*x.^2);
xx=-1:.01:1;
yy=1./(1+25*xx.^2);
yn=interp1(x,y,xx,'nearest');
yl=interp1(x,y,xx,'linear');
yc=interp1(x,y,xx,'cubic');
ys=interp1(x,y,xx,'spline');
plot(x,y,'o',xx,yy,xx,yn,xx,yl,xx,yc,xx,ys)
legend('data',...
       'function',...
       'nearest',...
       'linear',...
       'cubic',...
       'spline', 2);
max(abs(yn-yy))
max(abs(yl-yy))
max(abs(yc-yy))
max(abs(ys-yy))
```

Мы видим, что минимальную ошибку (0.02) дал кубический интерполянт, на втором месте — сплайн (0.06), далее — линейный интерполянт (0.07) и на последнем месте — ступенчатый интерполянт (0.31).

**Пример 14.** 1; $x=0:10$ ;  $y=\sin(x)$ ;  
 2; $xx=0:.25:10$ ;  
 3; $yn=interp1(x,y,xx,'nearest')$ ;  
 4; $yl=interp1(x,y,xx,'linear')$ ;

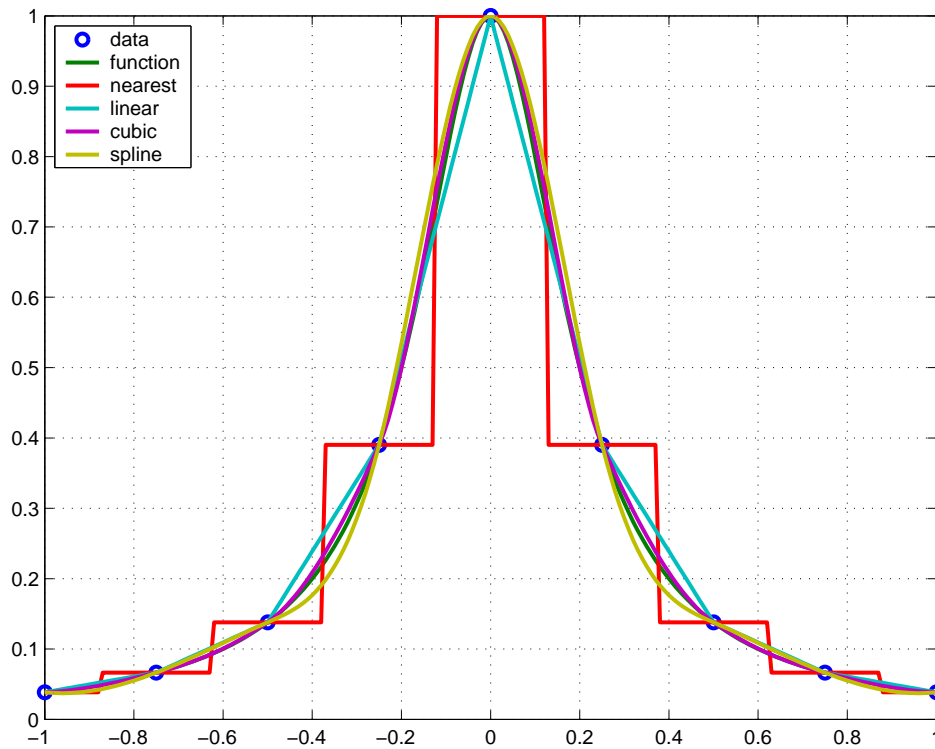


Рис. 7.4. Ступенчатая, линейная, кубическая интерполяция и интерполяция сплайном функции Рунге

```
5;yc=interp1(x,y,xx,'cubic');
6;ys=interp1(x,y,xx,'spline');
7;plot(x,y,'o',xx,yn,'g',xx,yl,'k',xx,yc,'r',xx,ys,'b'),grid;
8;legend('data',...
        'staiwice',...
        'linear',...
        'cubic',...
        'spline', 3);
```

% При равномерной решетке x можно использовать  
 % более быстрые методы 'со звездочкой':

```
6;ys=interp1(x,y,xx,'*spline');
```

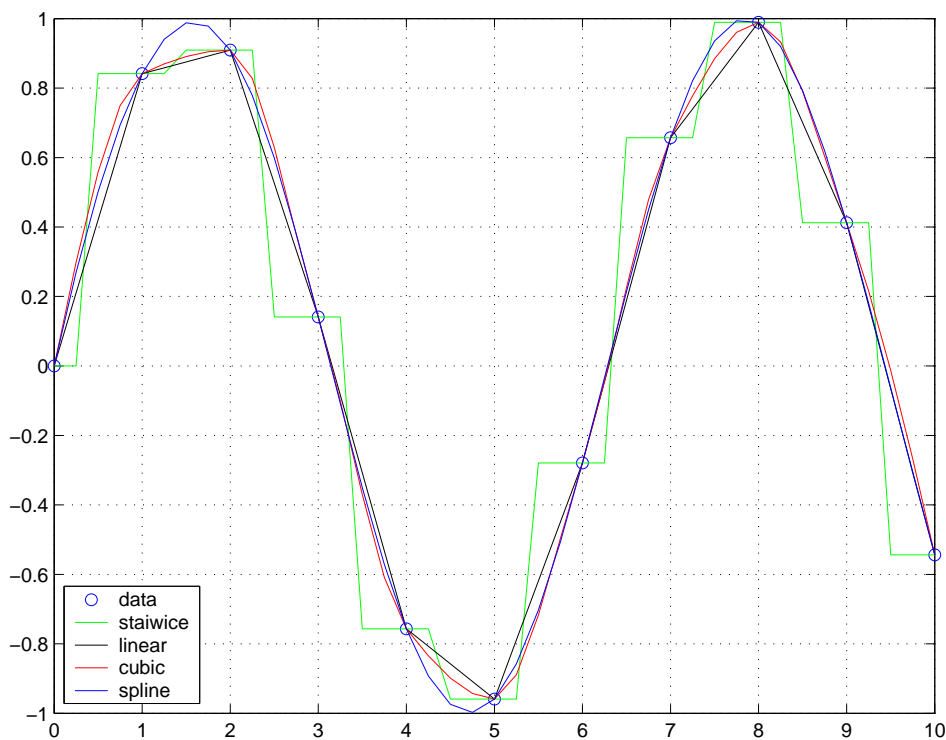


Рис. 7.5. Ступенчатая, линейная, кубическая интерполяция и интерполяция сплайном

#### 7.4. Кусочно-линейная интерполяция

Будем считать, что функция  $g(x)$  в точках  $x_1, x_2, \dots, x_n$  ( $x_1 < x_2 < \dots < x_n$ ) принимает значения  $y_1, y_2, \dots, y_n$  соответственно. Заменяя функцию  $g(x)$  на каждом из отрезков  $[x_i, x_{i+1}]$  линейной функцией  $l_i(x)$ , такой, что  $l_i(x_i) = y_i$ ,  $l_i(x_{i+1}) = y_{i+1}$  мы получим кусочно-линейный интерполянт

$$L(x) = l_i(x), \text{ если } x_i \leq x \leq x_{i+1}.$$

Нетрудно видеть, что

$$l_i(x) = y_i \frac{x - x_{i+1}}{x_i - x_{i+1}} + y_{i+1} \frac{x - x_i}{x_{i+1} - x_i}.$$

Если  $g(x)$  имеет непрерывную вторую производную, то погрешность интерполяции оценивается как

$$|L(x) - g(x)| < \frac{1}{8} h^2 \max |g''(x)| = O(h^2),$$

где  $h$  — максимальное расстояние между смежными узлами. Таким образом, выбрав достаточно много узлов, ошибку интерполяции можно сделать сколь угодно малой.

Кусочно-линейный интерполянт в системе MATLAB можно найти с помощью функции `interp1`. Если  $\mathbf{x}$  — массив содержащий узлы,  $\mathbf{y}$  — массив, содержащий значения функции  $g(x)$  в узлах,  $\mathbf{xx}$  — массив точек, то функция `interp1(x,y,xx)` возвращает набор значений интерполианта  $L(x)$  в точках  $\mathbf{xx}$ .

### 7.5. Эрмитов кубический интерполянт

Предположим, что в точках  $x_1, x_2, \dots, x_n$  ( $x_1 < x_2 < \dots < x_n$ ) известны не только значения функции  $y_1, y_2, \dots, y_n$   $g(x)$ , но и их производные  $d_1, d_2, \dots, d_n$ . Заменяя функцию  $g(x)$  на каждом из отрезков  $[x_i, x_{i+1}]$  кубическим многочленом  $c_i(x)$ , таким, что

$$\begin{aligned} c_i(x_i) &= y_i, & c_i(x_{i+1}) &= y_{i+1}, \\ c'_i(x_i) &= d_i, & c'_i(x_{i+1}) &= d_{i+1}, \end{aligned}$$

мы получим эрмитов кубический интерполянт

$$C(x) = c_i(x), \text{ если } x_i \leq x \leq x_{i+1}.$$

Очевидно, что эрмитов кубический интерполянт имеет непрерывную производную. Из дальнейшего будет следовать, что по значениям функции и ее первой производной в узлах, кубический интерполянт определяется единственным образом. Если  $g(x)$  имеет непрерывную четвертую производную, то погрешность интерполяции можно оценить как

$$|(x) - g(x)| < \text{const} \cdot h^4 \max |g^{(4)}(x)| = O(h^4),$$

где  $h$  — максимальное расстояние между смежными узлами. Кроме того, можно показать, что

$$|'(x) - g'(x)| = O(h^3), \quad |''(x) - g''(x)| = O(h^2), \quad |'''(x) - g'''(x)| = O(h).$$

Для вывода расчетных формул рассмотрим отрезок  $[0, 1]$  и условия

$$\begin{aligned} c(0) &= y_0, & c(1) &= y_1, \\ c'(0) &= d_0, & c'(1) &= d_1. \end{aligned}$$

Легко проверить, что единственным многочленом  $c(x)$  степени не выше 3, удовлетворяющим этим условиям, является многочлен

$$c(x) = a_0 x^3 + a_1 x^2 + a_2 x + a_3,$$

где

$$\begin{aligned}a_0 &= 2y_0 - 2y_1 + d_0 + d_1, \\a_1 &= -3y_0 + 3y_1 - 2d_0 - d_1, \\a_2 &= d_0, \\a_3 &= y_0.\end{aligned}$$

Чтобы перейти к произвольному отрезку  $[a, b]$  достаточно рассмотреть  $p(a + (b - a)x)$ .

Функция, которая находит pp-представление для кубического эрмита интерполянта, может выглядеть следующим образом:

```
function p=cubic(x,y,d);
n=length(x);
a=zeros(n-1,4);
for i=1:n-1
    a(i,1)=2*y(i)-2*y(i+1)+d(i)+d(i+1);
    a(i,2)=-3*y(i)+3*y(i+1)-2*d(i)-d(i+1);
    a(i,3)=d(i);
    a(i,4)=y(i);

    a(i,1)=a(i,1)/(x(i+1)-x(i))^3;
    a(i,2)=a(i,2)/(x(i+1)-x(i))^2;
    a(i,3)=a(i,3)/(x(i+1)-x(i))^1;
    a(i,4)=a(i,4);
end;
p=mkpp(x,a);
```

Приведем пример использования этой функции:

```
x=[0 1 2 3 4 5];
y=[0 1 1 2 3 3];
d0=[0 0 0 0 0 0];
d1=[1 1 1 1 1 1];
p0=cubic(x,y,d0);
p1=cubic(x,y,d1);
xx=0:.05:5;
yy0=ppval(p0,xx);
yy1=ppval(p1,xx);
plot(x,y,'o',xx,yy0,xx,yy1);
```

Если значения производных в узлах не известны, то можно попробовать заменить их приближениями, например:

$$d_i = \frac{\Delta_{i-1}h_i + \Delta_i h_{i-1}}{h_{i-1} + h_i}, \quad i = 1, \dots, n-1,$$

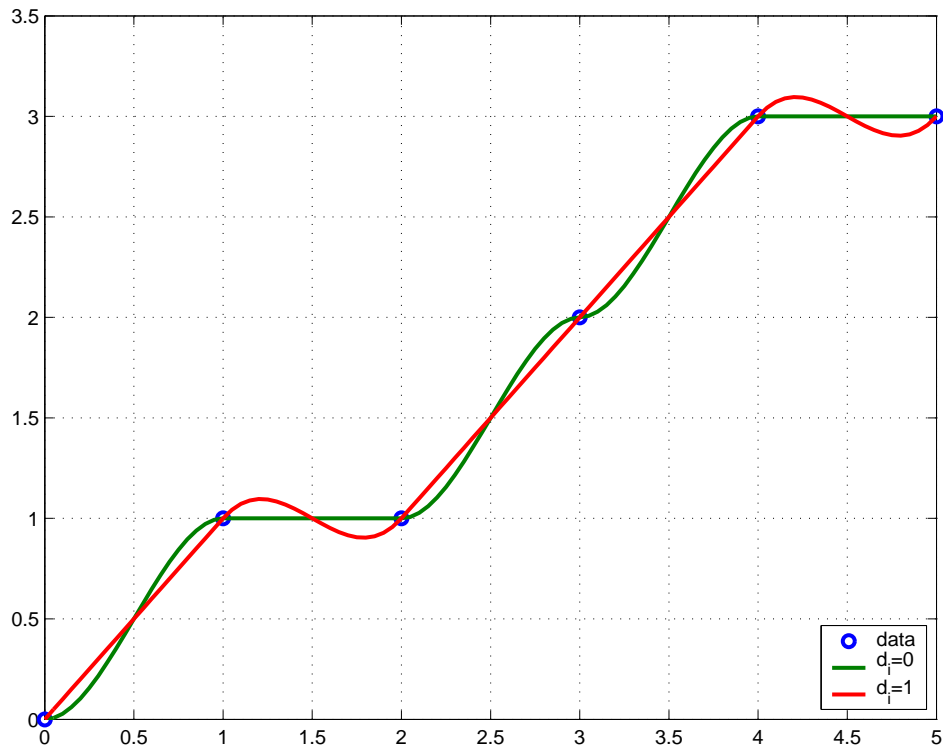


Рис. 7.6. Эрмитова кубическая интерполяция с явным заданием производных

$$d_0 = \frac{y_1 - y_0}{h_0}, \quad d_n = \frac{y_n - y_{n-1}}{h_{n-1}},$$

$$\text{где } \Delta_i = \frac{y_{i+1} - y_i}{h_i}, \quad h_i = x_{i+1} - x_i.$$

Реализуем предложенную схему приближения производных:

```
function d=deriv(x,y)
n=length(x);
v=diff(y);
h=diff(x);
d(1)=v(1)/h(1);
d(n)=v(n-1)/h(n-1);
for i=2:n-1
    d(i)=(v(i-1)*h(i)+v(i)*h(i-1))/(h(i-1)+h(i));
end;
```

Для наших данных график интерполянта слегка изменится:

```

x=[0 1 2 3 4 5];
y=[0 1 1 2 3 3];
d=deriv(x,y);
p=cubic(x,y,d);
xx=0:.05:5;
yy=ppval(p,xx);
plot(x,y,'o',xx,yy);

```

Сложный алгоритм выбора  $d_i$  реализован в стандартной функции `pchip`. Если  $x$ ,  $y$  — абсциссы и ординаты узлов интерполяции, то команда `уур=pchip(x,y,xx)` строит кубический интерполянт и возвращает его значения `уур` в точках `xx`. Построенный интерполянт обладает следующими особенностями:

- если на некотором отрезке данные монотонны, то монотонным будет сам интерполянт,
- если в некоторой точке данные имеют локальный оптимум, то локальный оптимум в той же точке будет и у интерполянта.

Проверим это

```

x=[0 1 2 3 4 5 6 7 8];
y=[0 1 1 2 3 4 3 2 1];
xx=0:.05:8;
уур=pchip(x,y,xx);
plot(x,y,'o',xx,уур);

```

## 7.6. Сплайны

Не всегда кубический эрмитов интерполянт может нас удовлетворить: в общем случае его вторая производная разрывна и, следовательно, график не достаточно гладок. Пусть, как и раньше, функция  $g(x)$  в точках  $x_1, x_2, \dots, x_n$  ( $x_1 < x_2 < \dots < x_n$ ) принимает значения  $y_1, y_2, \dots, y_n$  соответственно. На каждом из отрезков  $[x_i, x_{i+1}]$  заменим  $g(x)$  кубической функцией  $c_i(x)$ , такой, что  $c_i(x_i) = y_i$ ,  $c_i(x_{i+1}) = y_{i+1}$ . Кусочно-кубическая функция

$$S(x) = c_i(x), \text{ если } x_i \leq x \leq x_{i+1},$$

называется кубическим сплайном (или, просто, сплайном), если она везде имеет непрерывную вторую производную. Для того, чтобы сплайн



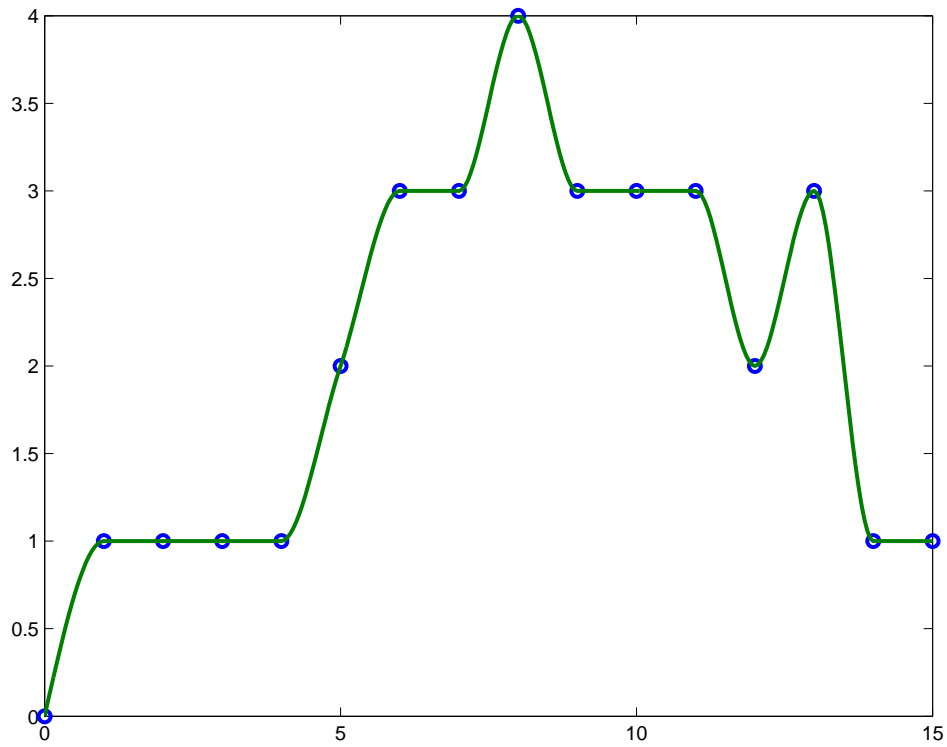


Рис. 7.7. Кубическая интерполяция, реализованная в системе МАТЛАВ

определялся единственным образом, зададим также значения первой производной в концах рассматриваемого интервала. Сплайн, определяемый условиями

$$S(x_0) = 0, \quad S(x_n) = 0$$

называется естественным.

Если  $g(x)$  имеет непрерывную четвертую производную, то погрешность интерполяции можно оценить как

$$|S(x) - g(x)| < \text{const} \cdot h^4 \max |g^{(4)}(x)| = O(h^4),$$

где  $h$  — максимальное расстояние между смежными узлами. Кроме того, можно показать, что

$$|S'(x) - g'(x)| = O(h^3), \quad |S''(x) - g''(x)| = O(h^2), \quad |S'''(x) - g'''(x)| = O(h).$$

Функция `yyp=spline(x,y,xx)` по значениям  $y$  в узлах  $x$  строит естественный сплайн и возвращает его значения `yyp` в точках  $xx$ . Эта функция выбирает такие значения первой производной в концевых точках, чтобы третья производная в узлах  $x_1$  и  $x_{n-1}$  была непрерывна.

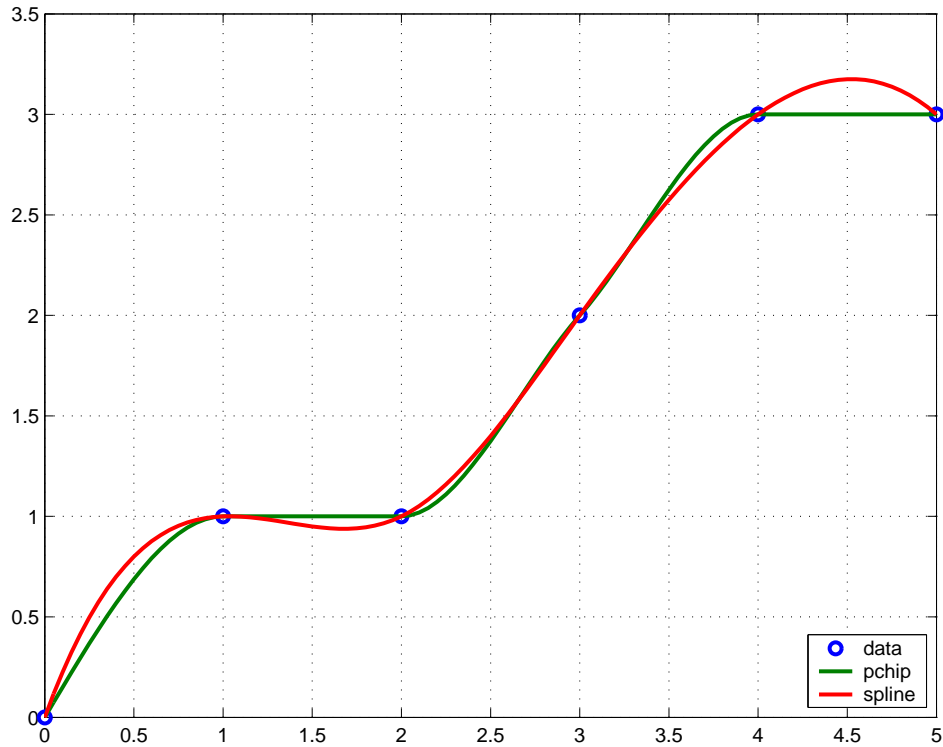


Рис. 7.8. Кубическая эрмитова интерполяция и кубический сплайн для монотонных данных

Несмотря на то, что графики сплайнов более гладкие нежели графики кубических интерполянтов, не всегда использование сплайнов дает удовлетворительные результаты.

**Пример 15.** Рассмотрим, например, случай, когда данные описывают некоторый естественный монотонный процесс:

```
x=[0 1 2 3 4 5];
y=[0 1 1 2 3 3];
xx=0:.05:5;
уур=pchip(x,y,xx);
уys=spline(x,y,xx);
plot(x,y,'o',xx,уур,xx,уys);
```

Сплайн оказался немонотонным, кубический интерполянт, который нашла функция `pchip`, — монотонным.

**Пример 16.** `load tolstoi`

```

hold on

m=length(Tolstoi1);
t=1:m;
tt=1:0.1:m;
sp=spline(t,Tolstoi1',tt)';
plot(sp(:,1),sp(:,2),'LineWidth',3); axis equal

m=length(Tolstoi2);
t=1:m;
tt=1:0.1:m;
sp=spline(t,Tolstoi2',tt)';
plot(sp(:,1),sp(:,2),'LineWidth',3); axis equal

m=length(Tolstoi3);
t=1:m;
tt=1:0.1:m;
sp=spline(t,Tolstoi3',tt)';
plot(sp(:,1),sp(:,2),'LineWidth',3); axis equal

m=length(Tolstoi4);
t=1:m;
tt=1:0.1:m;
sp=spline(t,Tolstoi4',tt)';
plot(sp(:,1),sp(:,2),'LineWidth',3); axis equal

```

### 7.7. Кривые Безье

Многочленами Бернштейна степени  $n$  называются многочлены вида

$$B_i^n(x) = \binom{n}{i} \frac{(x-a)^i (b-x)^{n-i}}{(b-a)^n}, \quad i = 0, \dots, n.$$

Например, для  $n = 3$  четыре многочлена Бернштейна имеют вид:

$$B_0^3 = \frac{(b-x)^3}{(b-a)^3}, \quad B_1^3 = \frac{(a-x)(b-x)^2}{(b-a)^3},$$

$$B_2^3 = \frac{(a-x)^2(b-x)}{(b-a)^3}, \quad B_3^3 = \frac{(a-x)^3}{(b-a)^3}.$$

Для вычисления многочленов Бернштейна удобно применять рекуррентные формулы:

$$\begin{aligned} B_0^0(x) &= 1, \\ B_{-1}^n(x) &= 0, \\ B_i^n(x) &= \frac{(x-a)B_{i-1}^{n-1} + (b-x)B_i^{n-1}}{b-a}, \quad 0 \leq i \leq n, \\ B_{n+1}^n(x) &= 0. \end{aligned}$$

Далее мы предполагаем, что  $a = 0$ ,  $b = 1$ . Этот случай, как мы увидим, очень важен для приложений. Функция для нахождения значений многочленов Бернштейна выглядит следующим образом:

```
function B=brnshtn(n,x)
% Bernshtein's polinomials [0, 1]
% B=brnshtn(n,x) returns matrix B of size m*(n+1)
% where B(i,j) is the value of the j-th Bernshtein's polynomial
% of degree n in the point x(i). x should be a vector
% of length m

[m,col]=size(x);
if min(m,col) ~=1
    error('x should be a vector');
end;
if col~=1
    x=x';
end;
B=zeros(m,n+1);

if n==0
    B(:,1)=1;
    return;
end;

for c=1:n
    if c==1
        B(:,c+1)=x;
    else
        B(:,c+1)=x.*B(:,c);
    end;

    for r=c:-1:2
```

```

    B(:,r)=x.*B(:,r-1)+(1-x).*B(:,r);
end;

if c==1
    B(:,1)=1-x;
else
    B(:,1)=(1-x).*B(:,1);
end;
end;

```

Легко проверить, что многочлены Бернштейна обладают свойством:

$$\begin{aligned} B_0^n(a) &= 1, & B_0^n(b) &= 0, \\ B_n^n(a) &= 0, & B_n^n(b) &= 1, \end{aligned} \quad (1)$$

$$\sum_{i=0}^n B_i^n(x) = 1, \quad i = 0, 1, \dots, 1. \quad (2)$$

Проверим это:

```

x=[0:.02:1];
b=brnshtn(4,x);
sum(b')
plot(x,b)
b=brnshtn(15,x);
sum(b')
plot(x,b)

```

Обратим внимание на максимумы функций. Из графиков видно, что абсцисса максимума растет с ростом номера многочлена. Для первой и последней функции максимум равен 1, тогда как для остальных он меньше.

Покажем теперь, как многочлены Берштейна можно применять для построения гладких плоских и пространственных кривых. Пусть про некоторую кривую известно, что она должна проходить вблизи точек  $\mathbf{p}_i$  ( $i = 0, \dots, n$ ). Рассмотрим тогда *векторную* функцию

$$\mathbf{B}(x) = \sum_{i=0}^n \mathbf{p}_i B_i^n(x), \quad x \in [0, 1].$$

Эта функция является параметрическим уравнением кривой Безье, очень удачно “огibaющей” заданные точки  $\mathbf{p}_i$ : из условий (1) следует, что кривая проходит через точки первую и последнюю точки, а из условия (2)

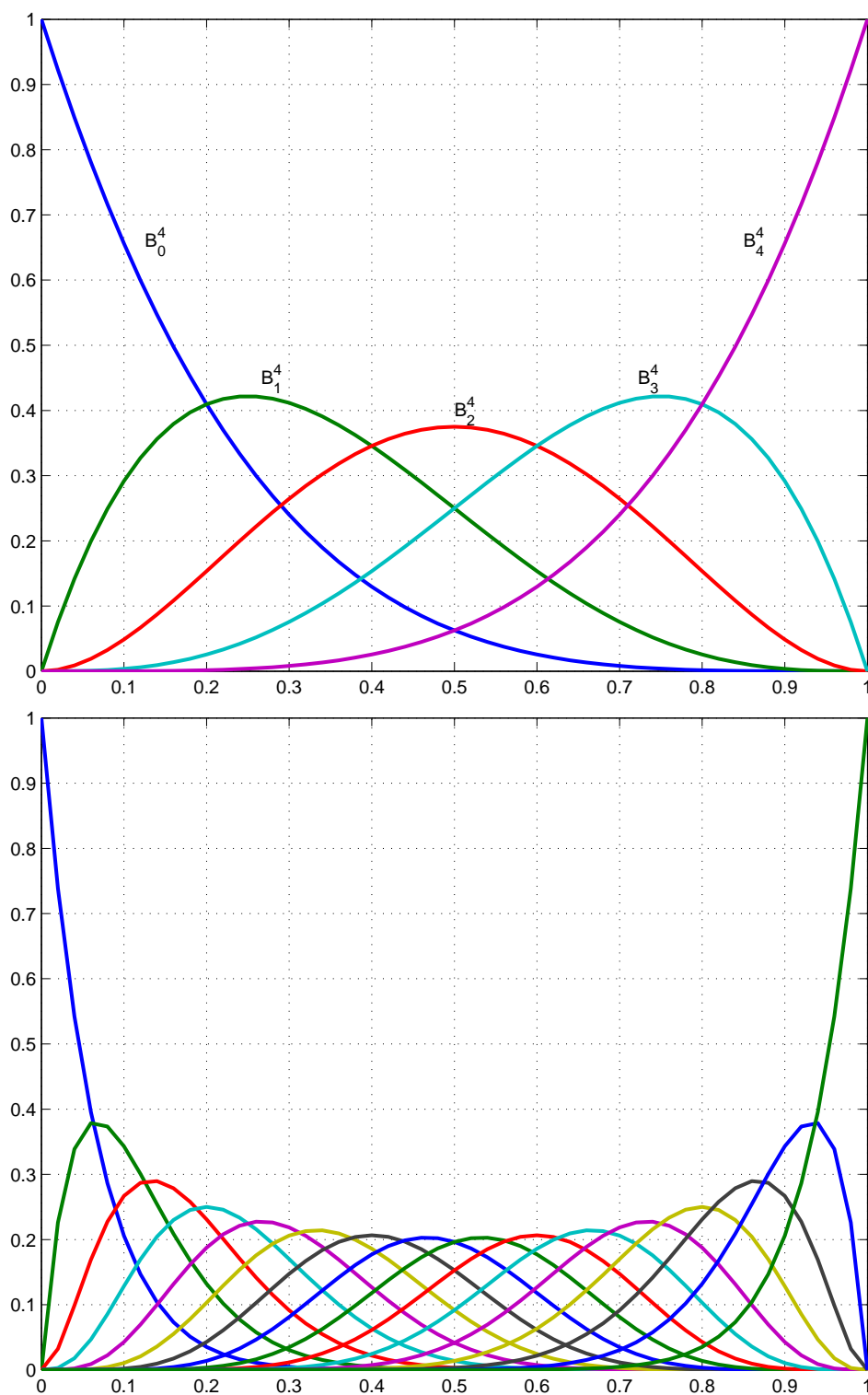


Рис. 7.9. Многочлены Бернштейна 4-ой и 15-ой степени

следует, что кривая полностью лежит внутри выпуклой оболочки всех заданных точек. Кроме того, графики многочленов Бернштейна подсказывают нам, что когда  $x$  близко к точке максимума  $i$ -го полинома Бернштейна, основную роль в правой части равенства (3) играет точка  $\mathbf{p}_i$ , т.е. кривая проходит около этой точки  $\mathbf{p}_i$ .

**Пример 17.** Построим кривую Безье для четырех наборов точек:

```
b=brnshtn(3,[0:0.01:1]);

x=[0 0; 1 -1; 2 1; 3 0];
p=b*x;
subplot(2,2,1);
plot(x(:,1),x(:,2),'o',p(:,1),p(:,2),'-')
text(x(:,1),x(:,2),[' 1';' 2';' 3';' 4'])
set(gca,'XTick',[],'YTick',[]);

x=[0 0; 2 1; 1 -1; 3 0];
p=b*x;
subplot(2,2,2);
plot(x(:,1),x(:,2),'o',p(:,1),p(:,2),'-')
text(x(:,1),x(:,2),[' 1';' 2';' 3';' 4'])
set(gca,'XTick',[],'YTick',[]);

x=[0 0; 3 3; 0 3; 3 0];
p=b*x;
subplot(2,2,4);
plot(x(:,1),x(:,2),'o',p(:,1),p(:,2),'-')
text(x(:,1),x(:,2),[' 1';' 2';' 3';' 4'])
set(gca,'XTick',[],'YTick',[]);

x=[1 0; 3 1; 0 1; 2 0];
p=b*x;
subplot(2,2,3);
plot(x(:,1),x(:,2),'o',p(:,1),p(:,2),'-')
text(x(:,1),x(:,2),[' 1';' 2';' 3';' 4'])
set(gca,'XTick',[],'YTick',[]);
```

## 7.8. В-сплайны

$$x(t) = a_3 t^3 + a_2 t^2 + a_1 t + a_0, \quad t \in [0, 1]$$

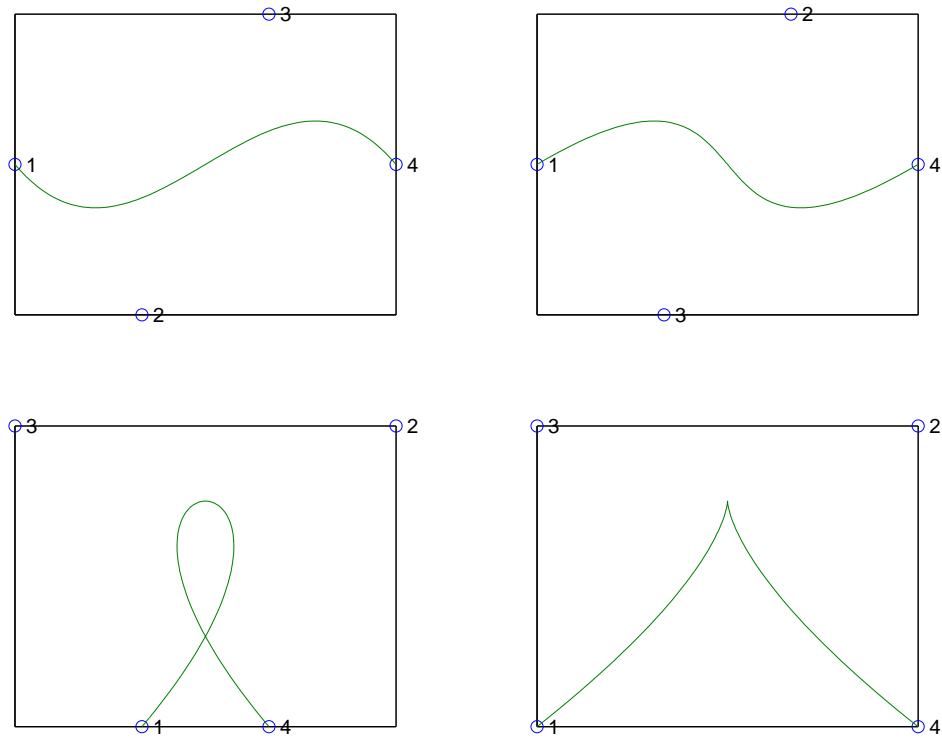


Рис. 7.10. Кривые Безье для заданного набора точек

$$\begin{aligned}
 a_3 &= (-x_{i-1} + 3x_i - 3x_{i+1} + x_{i+2})/6 \\
 a_2 &= (x_{i-1} - 2x_i + x_{i+1})/2 \\
 a_1 &= (-x_{i-1} + x_{i+1})/2 \\
 a_0 &= (x_{i-1} + 4x_i + x_{i+1})/6
 \end{aligned}$$

```

function B = b_spline(A,mm)
%function B = b_spline(A,mm)
%B-spline for curves

[m,n]=size(A);

p=zeros(4,n); %polynomials coefficients

B = [];

for i=2:m-2
    p(1,:) = (A(i-1,:)+4*A(i,:)+A(i+1,:))/6;

```



```

p(2,:) = (-A(i-1,:)+A(i+1,:))/2;
p(3,:) = (A(i-1:)-2*A(i,:)+A(i+1,:))/2;
p(4,:) = (-A(i-1:)+3*A(i:)-3*A(i+1:)+A(i+2:))/6;

for tt = [0:1/mm:1-1/mm]
    B = [B; ((p(4,:)*tt+p(3,:))*tt+p(2,:))*tt+p(1,:)];
end;
end;

```

**Пример 18.** spoon=[ %see also spoon.dat

```

.75 1.4
.5 1
.75 .6
1.5 .4
2.25 .7
2.5 .9
3 .85
5 .75
6.25 .8
6.5 .85
6.5 1
6.5 1.15
6.25 1.2
5 1.25
3 1.15
2.5 1.1
2.25 1.3
1.5 1.6
.75 1.4
.5 1
.75 .6];
bsp=b_spline(spoon,10);
[m,n]=size(spoon);
t=1:m; tt=1:0.1:m;
sp=spline(t, spoon',tt)';
plot(spoon(:,1), spoon(:,2), 'or', bsp(:,1), bsp(:,2), ...
     sp(:,1), sp(:,2)); axis equal
legend('points','b-spline','spline')
Другой вариант:
function p = b_spline(y)

```

```

n=length(y);

for i=2:n-2
    a(i-1,1)=(-y(i-1)+3*y(i)-3*y(i+1)+y(i+2))/6;
    a(i-1,2)=(y(i-1)-2*y(i)+y(i+1))/2;
    a(i-1,3)=(-y(i-1)+y(i+1))/2;
    a(i-1,4)=(y(i-1)+4*y(i)+y(i+1))/6;
end;

p=mkpp(1:n-2,a);

load spoon.dat
p1=b_spline(spoon(:,1));
p2=b_spline(spoon(:,2));
xx=1:0.1:19;
xx=1:0.1:19;
yy1=ppval(p1,xx);
yy2=ppval(p2,xx);
plot(yy1,yy2)

```

**Пример 19.**  $A=[$

```

0.36 0.91
0.36 0.91
0.36 0.91
0.30 0.93
0.22 0.87
0.22 0.78
0.31 0.60
0.29 0.51
0.23 0.46
0.14 0.45
0.10 0.51
0.10 0.51
0.10 0.51
];

```

```

m=length(A);
bsp=b_spline(A,10);
t=1:m;
tt=1:0.1:m;
sp=spline(t,A',tt)';
plot(A(:,1),A(:,2),'o',sp(:,1), sp(:,2),bsp(:,1), bsp(:,2),'LineWidth',2); axis equ

```

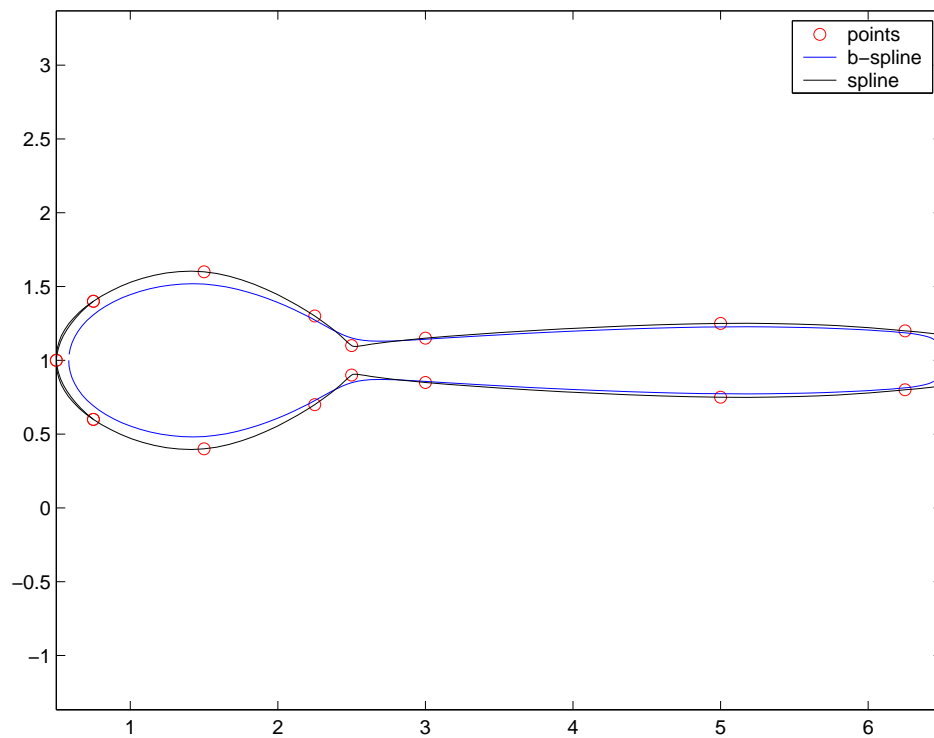


Рис. 7.11. Сплайн и В-сплайн для набора точек

### 7.9. Метод наименьших квадратов

**Пример 20.** `time = zeros(0,2);`  
`n = 100000;`  
`h = 5000;`  
`N = 2000000;`  
`for m = n:h:N,`  
`m,`  
`a = rand(1,m);`  
`tic;`  
`sort(a);`  
`time = [time; m, toc];`  
`end;`  
  
`save -ascii time.dat time`

```

load sorttime.dat
time = sorttime;

x = time(:,1);
y = time(:,2);

% x * log x
p = polyfit(x .* log(x), y, 1);
yy = polyval(p, x .* log(x));
err = sqrt(sum((yy - time(:, 2)) .^ 2))
% This is the same as F.normr where F is such that
% [p, F] = polyfit(x .* log(x), y, 1);

plot(x, y, '.', x, yy, 'r', 'LineWidth', 2)
title('x log x');
pause

% Further...

% linear function
[p, F] = polyfit(x, y, 1)
yy = polyval(p, x);
plot(x, y, '.', x, yy, 'm', 'LineWidth', 2)
title('ax+b')
pause

% quadratic function
[p, F] = polyfit(x, y, 2)
yy = polyval(p, x);
plot(x, y, '.', x, yy, 'b', 'LineWidth', 2)
title('ax^2+bx+c')
pause

% a x^b
[p, F] = polyfit(log(x), log(y), 1)
yy = exp(polyval(p, log(x)));
plot(x, y, '.', x, yy, 'k', 'LineWidth', 2)
title('ax^b')
pause

```

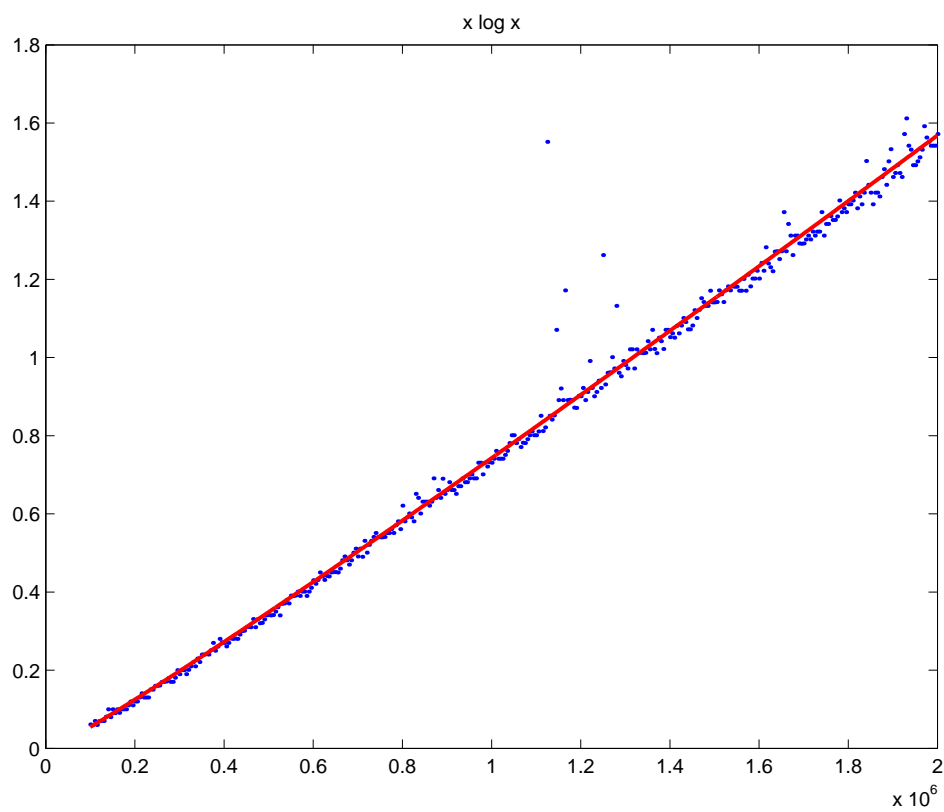


Рис. 7.12. Аппроксимация методом наименьших квадратов: подбор функции  $ax \log x + b$

```
% Pay attention to calculated values of p(0)
% --- They are the highest members of appr. polinomials
% and they are very small
```

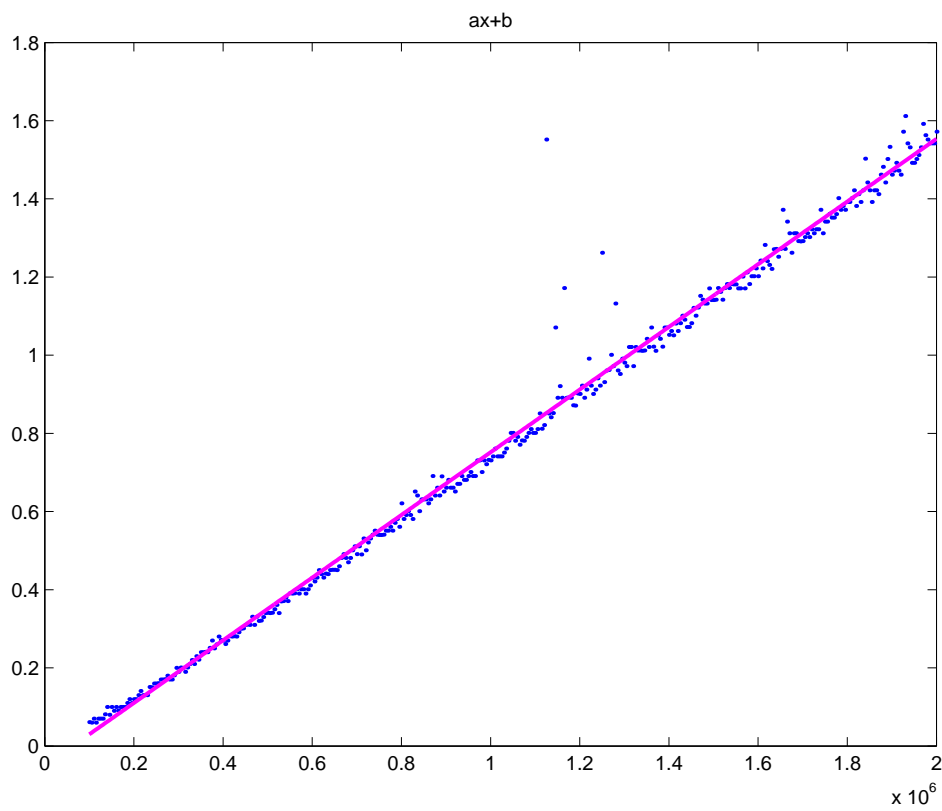


Рис. 7.13. Аппроксимация методом наименьших квадратов: подбор функции  $ax + b$

## Глава 8

# Анализ данных ?

### 8.1. Преобразование Фурье

$$X(\omega) = \int_{-\infty}^{+\infty} x(t)e^{-2\pi i t \omega} dt,$$

$$x(t) = \int_{-\infty}^{+\infty} X(\omega)e^{2\pi i t \omega} dt$$

$$X_p = \sum_{k=0}^{n-1} x_k e^{-2\pi i k p / n}, \quad (p = 0, \dots, n-1).$$

$$x_k = \frac{1}{n} \sum_{p=0}^{n-1} X_p e^{2\pi i k p / n}, \quad (k = 0, \dots, n-1).$$

```
x=1:.02:2;
```

```
function X = ft(x)
% FT (Slow) discrete Fourier transformation
% previous version
%
%      X=ft(x) calculates Fourier transformation for row x

n=length(x);
w=exp(2*pi*i/n);
k=0:n-1;
for p=0:n-1,
    X(p+1)=x*(w.^(~p*k'));
end
```

```

function x = ift(X)
% IFT (Slow) inverse discrete Fourier transformation
% previous version
%
%      x=ift(X) calculates inverse Fourier transformation for row X

n=length(X);
w=exp(2*pi*i/n);
p=0:n-1;
for k=0:n-1,
    x(k+1)=X*(w.^(k*p'))/n;
end;

x=sin(1:100)+rand(1,100);
subplot(1,3,1);
plot(x);

X=ft(x);
subplot(1,3,2);
plot(abs(X));

xnew=ift(X);
subplot(1,3,3);
plot(real(xnew));

max(abs(imag(xnew)))
norm(x-xnew)

n=length(x);
w=exp(2*pi*i/n);
k=0:n-1;
F=w.^(k'*k);
X=x*F;

1;ht=.001;t=0:ht:6;k1=50;k2=150;x=sin(2*pi*k1*t)+sin(2*pi*k2*t);
2;plot(x)
3;v=1:200;plot(x(v))
4;n=512;t=0:ht:6;k1=50;k2=150;A=0;x=sin(2*pi*k1*t)+sin(2*pi*k2*t)+2*A*(rand(1,100));
5;n=4096;X=fft(x,n);Pxx=X.*conj(X)/n;f=(0:n/2-1)/(ht*n);plot(f,Pxx(1:n/2))

1;ht=.001;t=0:ht:6;k1=50;k2=150;A=0;x=sin(2*pi*k1*t)+sin(2*pi*k2*t)+2*A*(rand(1,100));
2;v=1:200;plot(x(v))

```



```
3;n=4096;X=fft(x,n);Pxx=X.*conj(X)/n;f=(0:n/2-1)/(ht*n);plot(f,Pxx(1:n/2))

load gong;
Hz=11025;
plot(y);
pause;
plot(y(1:200));
pause;
n=pow2(nextpow2(length(y))-1)
tic;Y=fft(y,n);toc
PY=Y.*conj(Y)/n;
f=(0:n/2-1)*Hz/(n);
plot(f,PY(1:n/2));
```

## Программирование

### 9.1. М-файлы

Программой на языке МАТЛАВ мы будем называть последовательность команд, записанную в файле. Чтобы система принимала программу, у файла должно быть расширение `.m`, поэтому программы часто называют `m`-файлами. Программы вызываются по имени файла (без расширения). При первом обращении к программе она ищется на диске. В первую очередь поиск производится в текущем каталоге. Сменить текущий каталог можно командой

```
cd имя_каталога
```

или с помощью меню. Программу можно подготовить во внешнем редакторе (например, блокноте Windows), а можно воспользоваться встроенным редактором-отладчиком (Editor-Debugger). Для его вызова воспользуйтесь пунктом меню `FILE|NEW|M-FILE` или командой `edit`}.

После того, как программа найдена на диске, производится ее синтаксический анализ. Если в результате этого анализа обнаружены ошибки, информация о них выдается в рабочем окне. В случае успешного выполнения синтаксического анализа программы создается ее псевдокод (р-код), который загружается в рабочее пространство и выполняется. Если во время исполнения происходят ошибки, то сообщения о них также отражаются в командном окне.

При повторном обращении к программе, она будет найдена уже в рабочем пространстве и поэтому не потребуется времени на ее синтаксический анализ. Удалить псевдокод из рабочего пространства можно командой

```
clear имя_функции
```

В общем случае поиск очередного встретившегося в командах имени (это может быть имя переменной или программы) начинается с рабочего пространства. Если имя не найдено, то оно ищется среди встроенных (built-in) функций. Исходный код этих функций не доступен пользователю. Далее поиск продолжается в каталогах, записанных в списке доступа. Изменить эти пути можно либо через меню, либо с помощью команды `addpath`. Команда

```
addpath дир1 дир2 ... -begin
```

добавляет указанные директории в начало списка, а команда

```
addpath дир1 дир2 ... -end
```

добавляет директории в конец.

Команды в программе отделяются друг от друга так же, как и в командном окне: либо символом перехода на новую строку, либо знаками `;` , `—` — отличие двух последних такое же, как и в командном окне. Специальным образом обозначать конец программы никак не требуется. Внеочередное прекращение работы программы выполняется командой `return`. Символ `%` означает начало комментариев: все, что записано после него и до конца строки при синтаксическом анализе игнорируется. Все, что записано в первых строках-комментариях, автоматически подключается в систему справки и может быть вызвано с помощью команды

```
help имя_файла
```

По использованию переменных и оперативной памяти программы делятся на программы-сценарии и программы-функции.

## 9.2. Программы-сценарии

Программа-сценарий (script) — простейший тип программы. Программа-сценарий может использовать не только создаваемые ей самой переменные, но и использовать все переменные в рабочем пространстве. Созданные переменные так же располагаются в рабочем пространстве. Получить к ним доступ можно с помощью команды `keyboard`, которую нужно записать в программу. Эта команда передает управление в командное окно, в котором меняется вид приглашения:

```
K>>
```

Теперь в окне можно выполнять любые действия, в том числе просматривать значения переменных и изменять их. Чтобы продолжить выполнение программы необходимо выполнить команду `return`.

### 9.3. Программы-функции

После возможных пустых строк и строк, содержащих только комментарии, первая строка программы-функции должна иметь вид

```
function [y1,y2,...,ym]=ff(x1,x2,...,xn)
```

где **ff** — имя программы-функции (оно должно совпадать с именем файла), **x1, x2, ..., xn** — входные формальные параметры, **y1, y2, ..., yn** — выходные формальные параметры. Эту строку мы будем называть заголовком функции. У функции может быть только один выходной формальный параметр. В этом случае квадратные скобки в заголовке функции можно не писать. Формальные параметры (входные и выходные) используются в функции как локальные переменные. Конечно же, функция может создавать и использовать другие локальные переменные, которые, как и обычно, объявлять специальным образом не нужно.

Вызов программы осуществляется командами

```
[u1,y2,...,ul]=ff(v1,v2,...,vk)
```

где **v1, v2, ..., vk** и **u1, u2, ..., ul** — фактические входные и выходные параметры функции. При вызове функции фактические входные параметры засылаются по порядку в соответствующие выходные формальные параметры. После того, как функция завершила свою работу, формальные выходные параметры засылаются в фактические выходные параметры. Количество фактических параметров должно быть не больше количества формальных параметров, но может с ним и не совпадать. Это удобно, если используются значения аргументов по умолчанию. Количество фактических входных параметров и фактических выходных параметров, с которыми была вызвана функция всегда доступно в функции с помощью переменных **nargin** (количество входных параметров), **nargout** (количество выходных параметров).

Еще один способ вызова функции — это использование ее имени внутри выражения, например: **a=ff(k,2).^2**. Здесь первый выходной параметр функции возводится в квадрат и результат присваивается переменной **a**. Заметим, что количество выходных параметров функции может быть больше, но остальные аргументы при таком обращении к функции теряются. Если выражение состоит только из одного имени функции:

```
ff(v1,v2,...,vk)
```

то первый выходной параметр присваивается переменной **ans**.

```
function r = rank(A,tol)
%RANK    Matrix rank.
%   RANK(A) provides an estimate of the number of linearly
%   independent rows or columns of a matrix A.
%   RANK(A,tol) is the number of singular values of A
%   that are larger than tol.
%   RANK(A) uses the default tol = max(size(A)) * norm(A) * eps.

%   Copyright 1984-2000 The MathWorks, Inc.
%   $Revision: 5.9 $   $Date: 2000/06/01 02:04:15 $

s = svd(A);
if nargin==1
    tol = max(size(A)') * max(s) * eps;
end
r = sum(s > tol);
```

После того, как функция выполнила свою работу и произошло присваивание формальных выходных параметров фактическим, все локальные переменные функции удаляются. Переменные можно открыть для использования в командном окне, скриптах и других функциях с помощью команды

```
global v1,v2,...,vk
```

Это команда должна появиться во всех функциях, которые хотят использовать одни и те же переменные, упомянутые в списке. Для получения доступа к переменным функции из командного окна (или программы-сценария) необходимо выполнить эту команду в программе-сценарии или в командной строке.

### 9.3.1. Подфункции

M-файл с программой-функцией может содержать описание нескольких функций. Имя первой функции должно совпадать с именем файла. Только эта функция (основная) доступна извне. Остальные функции будем называть подфункциями. Каждая из них может быть вызвана либо из основной функции, либо из другой подфункции того же самого m-файла.

```
function [avg,med] = newstats(u) % Primary function
% NEWSTATS Find mean and median with internal functions.
n = length(u);
```

```
avg = mean(u,n);
med = median(u,n);
function a = mean(v,n)           % Subfunction
% Calculate average.
a = sum(v)/n;
function m = median(v,n)        % Subfunction
% Calculate median.
w = sort(v);
if rem(n,2) == 1
    m = w((n+1)/2);
else
    m = (w(n/2)+w(n/2+1))/2;
end
```

#### 9.4. Частные функции

Частные функции — это функции, размещенные в каталоге с именем **private**. Частные функции доступны из функций, расположенных в родительском каталоге. Этот каталог не следует указывать в путях доступа.

## Библиотека символьной математики

В данном разделе мы познакомимся с возможностями пакета символьных вычислений — Symbolic Math Toolbox. Данный пакет использует ядро системы Maple. Поддерживаются символьные вычисления в следующих областях математики:

- 1) математический анализ (производные, интегралы, пределы, суммы, ряды Тейлора)
- 2) линейная алгебра (системы линейных уравнений, обращение матриц, определитель, сингулярное разложение)
- 3) решение алгебраических уравнений
- 4) решение дифференциальных уравнений
- 5) специальные функции
- 6) вычисления с произвольной точностью

### 10.1. Создание символьных переменных

```
x = sym('x')
a = sym('alpha')

a = sym('a')
b = sym('b')
c = sym('c')
x = sym('x')
```

```
syms a b c x

rho = sym('(1 + sqrt(5))/2')
f = rho^2 - rho - 1
simplify(f)

f = sym('a*x^2 + b*x + c')
solve(f)
```

## 10.2. Символьно-числовые преобразования

Для перевода обычного числового значения (double) в символьную константу используют команду `sym`:

```
S = sym(A,flag)
```

где аргумент `flag` может принимать одно из следующих значений: `f`, `r`, `e` or `d`. Значение по умолчанию: `'r'`.

```
t = 0.1
sym(t,'f')
sym(t,'r')
sym(t,'e')
```

```
A = hilb(3)
A = sym(A)
```

### 10.2.1. Создание абстрактных функций

```
f = sym('f(x)')
df = (subs(f,'x','x+h') - f) / 'h'
```

### 10.2.2. Вызов функций Maple

```
kfac = sym('k!')
syms k n
subs(kfac,k,6), subs(kfac,k,n)
```

### 10.2.3. Создание символьных матриц

```
syms a b c
A = [a b c; b c a; c a b]
syms alpha beta;
A(2,3) = beta;
A = subs(A,b,alpha)
```



### 10.3. Создание символьных математических функций

Существует два способа создания символьных математических функций:

- Использование символьного выражения
- Создание m-файла

Примеры

```
syms x y z
r = sqrt(x^2 + y^2 + z^2)
t = atan(y/x)
f = sin(x*y)/(x*y)

function z = sinc(x)
%SINC The symbolic sinc function
%      sin(x)/x. This function
%      accepts a sym as the input argument.
if isequal(x,sym(0))
    z = 1;
else
    z = sin(x)/x;
end
```

Для работы с этими выражениями вы можете использовать `diff`, `int`, `subs` и другие функции из пакета символьной математики

## 10.4. Математический анализ

### 10.4.1. Производные

```
syms a x
f = sin(a*x)
diff(f)
diff(f,a)

syms a x
A = [cos(a*x), sin(a*x); -sin(a*x), cos(a*x)]
diff(A)
```

$$x = r \cos \lambda \cos \phi, \quad y = r \cos \lambda \sin \phi, \quad z = r \sin \lambda$$

Вычислим Якобиан

$$J = \frac{\partial(x, y, z)}{\partial(r, \lambda, \phi)}$$

```
syms r l f
x = r*cos(l)*cos(f); y = r*cos(l)*sin(f); z = r*sin(l);
J = jacobian([x; y; z], [r l f])

detJ = simple(det(J))
```

Производная	Команда
$\frac{df}{dx}$	<code>diff(f)</code> или <code>diff(f,x)</code>
$\frac{df}{dx}$	<code>diff(f,a)</code>
$\frac{d^2f}{dx^2}$	<code>diff(f,b,2)</code>
$\frac{\partial(r,t)}{\partial(u,v)}$	<code>jacobian([r;t],[u,v])</code>

#### 10.4.2. Пределы

```
syms h n x
limit((cos(x+h) - cos(x))/h,h,0 )

limit( (1 + x/n)^n,n,inf )
```

$$\lim_{x \rightarrow 0} \frac{1}{x} = \infty, \quad \lim_{x \rightarrow 0+} \frac{1}{x} = +\infty, \quad \lim_{x \rightarrow 0-} \frac{1}{x} = -\infty$$

```
limit(1/x,x,0)
limit(1/x)
limit(1/x,x,0,'left')
limit(1/x,x,0,'right')
```

Предел	Команда
$\lim_{x \rightarrow 0} f(x)$	<code>limit(f)</code>
$\lim_{x \rightarrow a} f(x)$	<code>limit(f,x,a)</code>
$\lim_{x \rightarrow a} f(x)$	<code>limit(f,a)</code>
$\lim_{x \rightarrow a-0} f(x)$	<code>limit(f,x,a,'left')</code>
$\lim_{x \rightarrow a+0} f(x)$	<code>limit(f,x,a,'right')</code>

## 10.4.3. Интегрирование

```

int(f)
int(f,v)
int(f,a,b)
int(f,v,a,b)

syms x
f=log(x)
int(f)

syms x
k = sym('1/sqrt(2)');
f = exp(-(k*x)^2);
ezplot(f)
int(f,x,-inf,inf)

```

## 10.5. Упрощения и подстановки

## 10.5.1. Упрощения

```

syms x
f = x^3-6*x^2+11*x-6
g = (x-1)*(x-2)*(x-3)
h = x*(x*(x-6)+11)-6
pretty(f), pretty(g), pretty(h)

collect(f)
expand(f)
horner(f)
factor(f)

syms x;
n = (1:9)';
p = x.^n + 1;
f = factor(p);
[p, f]

N = sym(1);
for k = 2:11
    N(k) = 10*N(k-1)+1;
end

```

```
[N' factor(N')]
```

```
simplify
simple
```

### 10.5.2. Подстановки

```
subexpr
```

```
syms a x
s = solve(x^3+a*x+1)
pretty(s)
r = subexpr(s)
sigma
```

```
subs
```

```
syms a b c
A = [a b c; b c a; c a b];
[v,E] = eig(A)
```

```
v = subexpr(v,'S')
```

```
E = subs(E,S,'S')
```

```
subs(v,a,10)
```

```
a = 10; b = 2; c = 10;
subs(S)
```

```
syms a b c
subs(S,{a,b,c},{10,2,10})
```

```
syms t
M = (1-t^2)*exp(-1/2*t^2);
P = (1-t^2)*sech(t);
```

```
ezplot(M); hold on; ezplot(P)
```

```
T = -6:0.05:6;
MT = double(subs(M,t,T));
PT = double(subs(P,t,T));
```

```

plot(T,MT,'b',T,PT,'r-.')
title(' ')
legend('M','P')
xlabel('t'); grid

```

### 10.6. Ортогональные многочлены

Следующие функции реализуют распространенные скалярные произведения в пространстве функций:

$$(f, g) = \int_{-1}^1 f(x) \cdot g(x) dx, \quad (f, g) = \int_{-1}^1 \frac{f(x) \cdot g(x)}{\sqrt{1-x^2}} dx,$$

```

function p=sprod(f,g)
p=int(a*b,-1,1);

function p=chebprod(f,g)
syms x;
p=int(a*b/(1-x^2)^(1/2),-1,1);

```

Символьная переменная `t` почему-то в системе MATLAB ведет себя не совсем хорошо. Например, команды

```

syms t;
t+0
t-0
t*1
t/1

```

приводят к сообщению об ошибке. За другими символьными переменными автор такого странного поведения не замечал. Это bug.

Напишем функцию построения ортогональной системы функций по заданной системе (процесс ортогонализации Грама-Шмидта):

```

function B=grmshm(A,prod)

if nargin<2
    prod=@sprod;
end

n=length(A);

```

```

B=A;

for i=1:n
    S=0;
    for j=1:i-1
        if S==0
            S=feval(prod,B(i),B(j))/feval(prod,B(j),B(j))*B(j);
        else
            S=S+feval(prod,B(i),B(j))/feval(prod,B(j),B(j))*B(j);
        end;
    end
    if S~=0
        B(i)=B(i)-S;
    end;
end;

```

Применим процесс ортогонализации к системе  $1, x, \dots, x^6$ :

```

syms x;
A=[1 x x^2 x^3 x^4 x^5 x^6];

L=grmshm(A);
for k=1:7
    ezplot(L(k), [-1 1]);
    pause;
end;

C=grmshm(A,@chebprod);
for k=1:7
    ezplot(C(k), [-1 1]);
    pause;
end;

```

## *Глава 11*

# Графические команды и функции

Два уровня работы с графическими объектами:

- высокоуровневый интерфейс,
- дескрипторная графика.

### **11.1. Дескрипторная графика**

Свойства, общие для всех объектов:

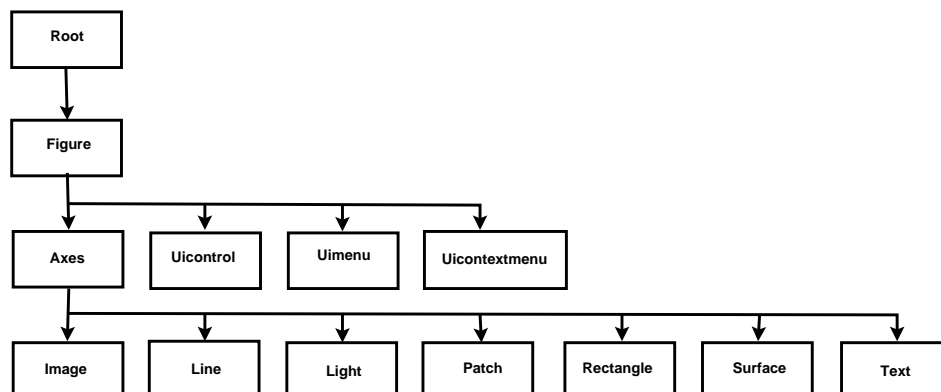


Рис. 11.1. Иерархия графических объектов

Property	Information Contained
BusyAction	Controls the way MATLAB handles callback routine interruption defined for the particular object
ButtonDownFcn	Callback routine that executes when button press occurs
Children	Handles of all this object's children objects
Clipping	Mode that enables or disables clipping (meaningful only for axes children)
CreateFcn	Callback routine that executes when this type of object is created
DeleteFcn	Callback routine that executes when you issue a command that destroys the object
HandleVisibility	Allows you to control the availability of the object's handle from the command line and from within callback routines
Interruptible	Determines whether a callback routine can be interrupted by a subsequently invoked callback routine
Parent	The object's parent
Selected	Indicates whether object is selected
SelectionHighlight	Specifies whether object visually indicates the selection state
Tag	User-specified object label
Type	The type of object (figure, line, text, etc.)
UserData	Any data you want to associate with the object
Visible	Determines whether or not the object is visible



### 11.2. Пример: создание графического объекта

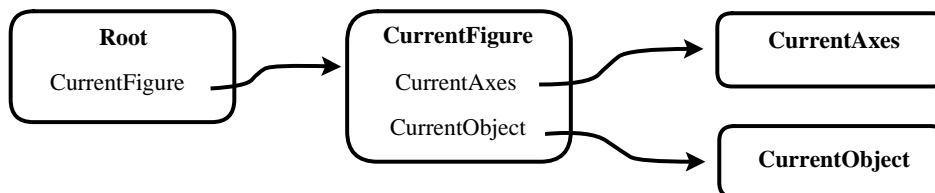
```

x=randn(200,1)-0.5;
y=randn(200,1)-0.5;
k = convhull(x,y);

fh = figure('Position',[350 275 400 300],'Color',[0.95 0.95 0.95]);
ah = axes('Color',[1 1 1],'XTick',[-3 -2 -1 0 1 2 3],...
          'YTick',[-3 -2 -1 0 1 2 3],...
          'DataAspectRatio',[1 1 1]);
ph = line('XData',x,'YData',y,...
          'LineStyle','none',...
          'Marker','.',...
          'MarkerSize',10,...
          'MarkerEdgeColor','b');
lh = line('XData',x(k),'YData',y(k),...
          'LineStyle','-','...',...
          'LineWidth',2,...
          'Color','r',...
          'Marker','.',...
          'MarkerSize',20,...
          'MarkerEdgeColor','r');

```

### 11.3. Текущие графические объекты



- `gcf` возвращает дескриптор `CurrentFigure`,
- `gca` возвращает дескриптор `CurrentAxes`,
- `gco` возвращает дескриптор `CurrentObject`.

### 11.4. Определение граней объекта `Patch`

Объект `Patch` предназначен для изображения многогранных или гладких поверхностей. В обоих случаях поверхность задается как на-

бор полигональных граней. В случае гладкой поверхности эти грани лишь аппроксимируют поверхность. Определить эти грани можно двумя способами:

- задать массивы `XData`, `YData`, `ZData`, определяющие координаты вершин каждой грани;
- задать массив `Vertices` с координатами вершин и массив `Faces` с информацией о том, как эти вершины соединяются в грани.

Для поверхностей с большим числом граней предпочтителен второй способ, так как он обычно требует меньшего объема памяти: одна и та же вершина, как правило, повторяется в нескольких гранях.

#### 11.4.1. Определение граней объекта `Patch` с помощью задания координат `XData`, `YData`, `ZData`

В первом способе нужно определить поля `XData`, `YData`, `ZData` объекта `Patch`. Эти поля должны представлять собой матрицы, в каждом столбце которых записаны координаты  $x$ ,  $y$ ,  $z$  соответственно вершин одной грани поверхности. Вершины должны появляться в направлении обхода грани (по или против часовой стрелке). Необходимо избегать граней с самопересекающейся границей. Такие грани следует разбивать на несколько частей. Чтобы представлять объекты, состоящие из граней с разным числом вершин, используйте `NaN` для обозначения фиктивной координаты.

Функция `patch(X,Y,Z,c)` изображает поверхность и возвращает ее дескриптор. Полям `XData`, `YData`, `ZData` присваиваются значения `X`, `Y`, `Z` соответственно. Полю `CData`, отвечающему за цвет поверхности, присваивается значение `c`. Подробнее о цвете мы поговорим ниже.

С помощью функции `patch(X,Y,Z,c,'Property',Value,...,'Property',Value)` можно задать дополнительные параметры объекта `Patch`.

Задать свойства `XData`, `YData`, `ZData` можно напрямую командой `patch('XData',X,'YData',Y,'ZData',Z,'FaceVertexCData',c)`.

**Пример 21.** Нарисуем единичный куб желтого ('y') цвета.

```
X=[0 0 0 0 0 1
    1 1 0 1 0 1
    1 1 0 1 1 1
    0 0 0 0 1 1];
```

```
Y=[0 0 0 0 1 0
```

```

0 0 1 0 1 1
1 0 1 1 1 1
1 0 0 1 1 0];

Z=[0 0 0 1 0 0
   0 0 0 1 1 0
   0 1 1 1 1 1
   0 1 1 1 0 1];

patch(X,Y,Z,'y')
axis equal
axis([-0.2 1.2 -0.2 1.2 -0.2 1.2])

```

#### 11.4.2. Определение граней объекта *Patch* с помощью задания массивов **Vertices** и **Faces**

Другой способ задания граней поверхности — определить поля **Vertices** и **Faces**. В матрице **Vertices** построчно записываются координаты каждой вершины поверхности, в матрице **Faces** построчно записываются индексы вершин, составляющих каждую грань поверхности. Вершины должны появляться в направлении обхода грани (по или против часовой стрелке).

**Пример 22.** Нарисуем куб бирюзового ('c') цвета.

```

V = [0 0 0
     1 0 0
     0 1 0
     1 1 0
     0 0 1
     1 0 1
     0 1 1
     1 1 1];

F = [1 2 4 3
     1 2 6 5
     1 3 7 5
     5 7 8 6
     3 4 8 7
     2 4 8 6];

axis equal;

```

```
axis off;
view(3)
patch('Vertices',V,'Faces',F,'FaceColor','c')
```

### 11.5. Стиль ребер и вершин

Управляя следующими свойствами объекта **Patch** вы можете задавать стиль и толщину ребер, тип и размер маркеров вершин.

Поле **LineStyle** позволяет задать один из следующих 5 стилей ребер:

Символ	Стиль линии
'_'	сплошная линия (по умолчанию)
'_.'	штрих-линия
'.'	пунктирная линия
'-.'	штрих-пунктирная линия
'none'	нет линии

Поле **LineWidth** позволяет задать толщину линий в пунктах (points): Пункт = 1/72дюйма  $\approx$  0.35мм. По умолчанию, **LineWidth** равно 0.5 пункта.

Поле **Marker** позволяет задать тип маркеров: '.', '+', 'x', '\*', 'o', '~', 'v', '<', '>', 's', 'd', 'p', 'h', 'none'.

Поле **MarkerSize** позволяет задать размер маркера в пунктах. По умолчанию, **MarkerSize** равно 6 пунктам. Заметим, что система МАТЛАВ отображает маркер '.' размером в 1/3 от заданного.

### 11.6. Цветовые характеристики объекта Patch

#### 11.6.1. Свойство FaceColor

Свойство **FaceColor** определяет видимость, цвет и способ заливки граней. Рассмотрим возможные значения этого поля:

RGB-вектор или одно из имен для обозначения цвета (по умолчанию, 'k') — все грани окрашиваются в один цвет,

'none' — грани не отображаются,

'flat' — каждая грань окрашивается в свой цвет, определяемый свойством **CData** или **FaceVertexCData** (см. ниже),

'interp' — заливка грани формируется линейной интерполяцией значений цвета в вершинах грани; цвет в вершинах определяется свойством **CData** или **FaceVertexCData** (см. ниже).

### 11.6.2. Свойство `EdgeColor`

Свойство `Edge` определяет видимость, цвет и способ заливки граней. Рассмотрим возможные значения этого поля:

RGB-вектор или одно из имен для обозначения цвета (по умолчанию, `'k'`) — все ребра окрашиваются в один цвет,

`'none'` — ребра не отображаются,

`'flat'` — каждое ребро окрашивается в свой цвет, определяемый свойством `CData` или `FaceVertexCData` (см. ниже),

`'interp'` — заливка ребра формируется интерполяцией значений цвета в вершинах ребра; цвет в вершинах определяется свойством `CData` или `FaceVertexCData` (см. ниже).

### 11.6.3. Свойство `MarkerEdgeColor`

Свойство `MarkerEdgeColor` определяет видимость и цвет границы маркеров, которые могут отображаться в вершинах объекта *Patch*. Рассмотрим возможные значения этого поля:

RGB-вектор или одно из имен для обозначения цвета границы маркера или всего маркера целиком (для незаполненных маркеров),

`'none'` — граница маркера или весь маркер (для незаполненных маркеров) не отображается,

`'auto'` (по умолчанию) — полю `MarkerEdgeColor` присваивается значение поля `EdgeColor`,

`'flat'` — граница маркера или весь маркер (для незаполненных маркеров) окрашивается в свой цвет, определяемый свойством `CData` или `FaceVertexCData` (см. ниже).

### 11.6.4. Свойство `MarkerFaceColor`

Свойство `MarkerEdgeColor` определяет видимость и цвет тела маркеров, которые могут отображаться в вершинах объекта *Patch*. Данное поле используется только для заполненных типов маркеров, так как для незаполненных маркеров все цветовые характеристики определяются значением поля `MarkerFaceColor`. Рассмотрим возможные значения поля `MarkerEdgeColor`:

RGB-вектор или одно из имен для обозначения цвета тела маркера,

`'none'` — внутренность маркера прозрачна,

`'auto'` (по умолчанию) — полю `MarkerEdgeColor` присваивается значение поля `Color` объекта *Axes* или объекта *Figure*, если значение `Color` объекта *Axes* равно `none`,

'flat' — тело каждого маркера окрашивается в свой цвет, определяемый свойством `CData` или `FaceVertexCData` (см. ниже).

#### 11.6.5. Высококачественные реалистичные изображения

Для получения высококачественных реалистичных изображений при задании цвета объекта `Patch` необходимо пользоваться свойствами `CData` или `FaceVertexCData`. Способ задания цветовых характеристик зависит от того, как были заданы грани объекта. Если при задании граней использовались свойства `XData`, `YData`, `ZData`, то основным свойством, которое управляет цветом, является свойство `CData`. Если при определении объекта использовались свойства `Faces`, `Vertices`, то для определения цветовых характеристик необходимо использовать свойство `FaceVertexCData`:

Вершины и грани	Цвет
<code>Faces</code> <code>Vertices</code>	<code>FaceVertexCData</code>
<code>XData</code> <code>YData</code> <code>ZData</code>	<code>CData</code>

В обоих случаях вы можете выбирать произвольные способы расцветки граней, ребер и вершин поверхности. В следующих разделах мы остановимся на этом подробнее.

Каждая грань окрашивается либо в один цвет (flat), либо заливка грани формируется интерполяцией значений цвета в вершинах этой грани (interpolated).

Каждое ребро окрашивается либо в один цвет (flat), либо заливка ребра формируется интерполяцией значений цвета в вершинах ребра (interpolated).

#### 11.6.6. Свойство `CData`

Свойство `CData` объекта `Patch` определяет цвет поверхности. Это свойство используется только тогда, когда для задания граней поверхности использовались поля `XData`, `YData`, `ZData`. С помощью этого свойства можно задать один цвет для всех граней и ребер, задать свой цвет для каждой грани и каждого ребра или определить способ заливки граней и ребер поверхности. Способ, согласно которому MATLAB интерпретирует значение `CData` зависит от типа данных, хранящихся в

этом поле. Предположим, что поверхность состоит из  $m$  граней и каждая грань —  $k$ -угольник. Следующая таблица показывает возможные способы интерпретации свойства `CData`:

X, Y, ZData	CData		Результат
Размеры	Indexed	True Color	
$k \times m$	скаляр	$1 \times 1 \times 3$	Один цвет для всего объекта
$k \times m$	$1 \times m$	$1 \times m \times 3$	Свой цвет для каждой грани
$k \times m$	$k \times m$	$k \times m \times 3$	Свой цвет для каждой вершины

Заметим, что возможен как относительный (indexed), так и абсолютный (true colors) способ задания цвета. Если используется относительный способ, то поле `CDataMapping` указывает, масштабируются ('scaled') или нет (direct) индексы палитры. По умолчанию поле `CDataMapping` имеет значение 'scaled'.

Остановимся подробнее на способах задания цвета:

- *Один цвет для всего объекта.* В этом случае значение `CData` — либо индекс в палитре, либо RGB-вектор.

Возможные значения для `FaceColor` и `EdgeColor`: 'none' или 'flat'.

Возможные значения для `MarkerFaceColor` и `MarkerEdgeColor`: 'none' или 'auto'.

- *Свой цвет для каждой грани.* В этом случае в массиве `CData` для каждой грани указывается либо индекс в палитре, либо RGB-вектор.

Возможные значения для `FaceColor` и `EdgeColor`: 'none' или 'flat'.

Возможные значения для `MarkerFaceColor` и `MarkerEdgeColor`: 'none', 'auto' или 'flat'.

- *Свой цвет для каждой вершины.* В этом случае в массиве `CData` для каждой вершины указывается либо индекс в палитре, либо RGB-вектор.

Возможные значения для `FaceColor` и `EdgeColor`: 'none', 'flat' или 'interp'.

- В случае 'none' грани (ребра) не отображаются.
- В случае 'flat' цвет грани (ребра) определяется цветом первой вершины на этой грани (ребре).

- В случае `'interp'` заливка грани (ребра) определяется интерполяцией значений цвета в вершинах этой грани (ребра).

Возможные значения для `MarkerFaceColor` и `MarkerEdgeColor`: `'none'`, `'auto'` или `'flat'`.

**Пример 23.** Нарисуем единичный куб с разноцветными гранями:

```
X=[0 0 0 0 0 1
    1 1 0 1 0 1
    1 1 0 1 1 1
    0 0 0 0 1 1];
```

```
Y=[0 0 0 0 1 0
    0 0 1 0 1 1
    1 0 1 1 1 1
    1 0 0 1 1 0];
```

```
Z=[0 0 0 1 0 0
    0 0 0 1 1 0
    0 1 1 1 1 1
    0 1 1 1 0 1];
```

```
c=1:6
```

```
patch(X,Y,Z,1:6)
axis square
view(3)
```

Теперь нарисуем куб, задавая случайным образом цвет вершин:

```
colormap autumn
patch(X,Y,Z,rand(4,6))
axis square
view(3)
```

Проиллюстрируем все 6 возможных способов интерпретации поля `FaceVertexCData`:

```
% cubes.m
```

```
X=[0 1 1 0 1 0
    0 1 0 1 1 0
```



```
0 1 0 1 0 1
0 1 1 0 0 1];

Y=[0 0 1 0 0 0
   1 1 1 0 1 1
   1 1 1 0 1 1
   0 0 1 0 0 0];

Z=[0 1 1 1 0 1
   0 1 1 1 0 1
   1 0 0 0 0 1
   1 0 0 0 0 1];

colormap('spring')

subplot(3,2,1);
axis equal;
axis off;
view(3)
patch(X, Y, Z, 1);

subplot(3,2,2);
axis equal;
axis off;
view(3)
patch(X, Y, Z, rand(1,1,3));

subplot(3,2,3);
axis equal;
axis off;
view(3)
patch(X, Y, Z, rand(1,6));

subplot(3,2,4);
axis equal;
axis off;
view(3)
patch(X, Y, Z, rand(1,6,3));

subplot(3,2,5);
axis equal;
```

```
axis off;
view(3)
patch(X, Y, Z, rand(4,6));

subplot(3,2,6);
axis equal;
axis off;
view(3)
patch(X, Y, Z, rand(4,6,3));
```

#### 11.6.7. Свойство FaceVertexCData

Свойство **FaceVertexCData** объекта **Patch** определяет цвет поверхности. Это свойство используется только тогда, когда для задания граней поверхности использовались поля **Faces**, **Vertices**. С помощью этого свойства можно задать один цвет для всех граней и ребер, задать свой цвет для каждой грани и каждого ребра или определить способ заливки граней и ребер поверхности. Способ, согласно которому МАТЛАВ интерпретирует значение **FaceVertexCData** зависит от типа данных, хранящихся в этом поле. Предположим, что поверхность состоит из  $m$  граней, имеет всего  $n$  вершин и каждая грань —  $k$ -угольник. Следующая таблица показывает возможные способы интерпретации свойства **FaceVertexCData**:

Vertices	Faces	FaceVertexCData		Результат
Размеры	Размеры	Indexed	True Color	
$n \times 3$	$m \times k$	скаляр	$1 \times 3$	Один цвет для всего объекта
$n \times 3$	$m \times k$	$m \times 1$	$m \times 3$	Свой цвет для каждой грани
$n \times 3$	$m \times k$	$n \times 1$	$n \times 3$	Свой цвет для каждой вершины

Заметим, что возможен как относительный (**indexed**), так и абсолютный (**true colors**) способ задания цвета. Если используется относительный способ, то поле **CDataMapping** указывает, масштабируются (**'scaled'**) или нет (**'direct'**) индексы палитры. По умолчанию поле **CDataMapping** имеет значение **'scaled'**.

Остановимся подробнее на способах задания цвета:

- *Один цвет для всего объекта.* В этом случае значение **FaceVertexCData** — либо индекс в палитре, либо RGB-вектор.

Возможные значения для **FaceColor** и **EdgeColor**: **'none'** или **'flat'**.

Возможные значения для `MarkerFaceColor` и `MarkerEdgeColor`: `'none'` или `'auto'`.

- *Свой цвет для каждой грани.* В этом случае в массиве `FaceVertexCData` для каждой грани указывается либо индекс в палитре, либо RGB-вектор.

Возможные значения для `FaceColor` и `EdgeColor`: `'none'` или `'flat'`.

Возможные значения для `MarkerFaceColor` и `MarkerEdgeColor`: `'none'`, `'auto'` или `'flat'`.

- *Свой цвет для каждой вершины.* В этом случае в массиве `FaceVertexCData` для каждой вершины указывается либо индекс в палитре, либо RGB-вектор.

Возможные значения для `FaceColor` и `EdgeColor`: `'none'`, `'flat'` или `'interp'`.

- В случае `'none'` грани (ребра) не отображаются.
- В случае `'flat'` цвет грани (ребра) определяется цветом первой вершины на этой грани (ребре).
- В случае `'interp'` заливка грани (ребра) определяется интерполяцией значений цвета в вершинах этой грани (ребра).

Возможные значения для `MarkerFaceColor` и `MarkerEdgeColor`: `'none'`, `'auto'` или `'flat'`.

**Пример 24.** Нарисуем единичный куб с разноцветными гранями:

```
V = [0 0 0
      1 0 0
      0 1 0
      1 1 0
      0 0 1
      1 0 1
      0 1 1
      1 1 1];
```

```
F = [1 2 4 3
      1 2 6 5
      1 3 7 5
      5 7 8 6
      3 4 8 7]
```

```

        2 4 8 6];

colormap winter
axis square;
axis off;
view(3);
c=(1:6)';
patch('Vertices',V,'Faces',F,'FaceVertexCData',c,'FaceColor','flat')

```

Воспользуемся тем, что в нашем примере элементы матрицы  $V$  заключены между нулем и единицей, поэтому она также может задавать цвет (абсолютный способ задания цвета — true colors). Нарисуем куб, задавая цвет в вершинах и используя интерполяцию на гранях:

```

patch('Vertices',V,'Faces',F,'FaceVertexCData',V,'FaceColor','interp')

```

## 11.7. Освещение

### 11.7.1. AmbientStrength

```

%lghtamb.m
[X, Y, Z]=sphere(50);
axis equal

surface(X,Y,Z,'FaceColor',[1 0 0],'EdgeColor','none','AmbientStrength',0);
surface(X+3,Y,Z,'FaceColor',[1 0 0],'EdgeColor','none','AmbientStrength',0.5);
surface(X+6,Y,Z,'FaceColor',[1 0 0],'EdgeColor','none','AmbientStrength',1);

camlight;
lighting phong;

```

### 11.7.2. DiffuseStrength

```

%lghtdif

[X, Y, Z]=sphere(50);
axis equal

surface(X, Y, Z, 'FaceColor', [1 0 0], 'EdgeColor', 'none', ...
'DiffuseStrength', 0);
surface(X+3, Y, Z, 'FaceColor', [1 0 0], 'EdgeColor', 'none', ...

```

```
    'DiffuseStrength', 0.5);
surface(X+6, Y, Z, 'FaceColor', [1 0 0], 'EdgeColor', 'none', ...
    'DiffuseStrength', 1);

camlight;
lighting phong;
```

### 11.7.3. SpecularStrength

```
%lightsexp

[X, Y, Z]=sphere(50);
axis equal

surface(X, Y, Z, 'FaceColor', [1 0 0], 'EdgeColor', 'none', ...
    'SpecularStrength', 0);
surface(X+3, Y, Z, 'FaceColor', [1 0 0], 'EdgeColor', 'none', ...
    'SpecularStrength', 0.5);
surface(X+6, Y, Z, 'FaceColor', [1 0 0], 'EdgeColor', 'none', ...
    'SpecularStrength', 1);

camlight;
lighting phong;
```

### 11.7.4. SpecularExponent

```
%lightsexp

[X, Y, Z]=sphere(50);
axis equal

surface(X, Y, Z, 'FaceColor', [1 0 0], 'EdgeColor', 'none', ...
    'SpecularExponent', 20);
surface(X+3, Y, Z, 'FaceColor', [1 0 0], 'EdgeColor', 'none', ...
    'SpecularExponent', 13);
surface(X+6, Y, Z, 'FaceColor', [1 0 0], 'EdgeColor', 'none', ...
    'SpecularExponent', 5);

camlight;
lighting phong;
```

**11.7.5. SpecularColorReflectance**

```
%lightscol

[X, Y, Z]=sphere(50);
axis equal

surface(X, Y, Z, 'FaceColor', [1 0 0], 'EdgeColor', 'none', ...
    'SpecularColorReflectance', 0);
surface(X+3, Y, Z, 'FaceColor', [1 0 0], 'EdgeColor', 'none', ...
    'SpecularColorReflectance', 0.7);
surface(X+6, Y, Z, 'FaceColor', [1 0 0], 'EdgeColor', 'none', ...
    'SpecularColorReflectance', 1);

camlight;
lighting phong;
```

**11.7.6. Материал**

Controls the reflectance properties of surfaces and patches

Syntax material shiny material dull material metal material([ka kd ks])  
 material([ka kd ks n]) material([ka kd ks n sc]) material default Description  
 material sets the lighting characteristics of surface and patch objects.

material shiny sets the reflectance properties so that the object has a high specular reflectance relative the diffuse and ambient light and the color of the specular light depends only on the color of the light source.

material dull sets the reflectance properties so that the object reflects more diffuse light, has no specular highlights, but the color of the reflected light depends only on the light source.

material metal sets the reflectance properties so that the object has a very high specular reflectance, very low ambient and diffuse reflectance, and the color of the reflected light depends on both the color of the light source and the color of the object.

material([ka kd ks]) sets the ambient/diffuse/specular strength of the objects.

material([ka kd ks n]) sets the ambient/diffuse/specular strength and specular exponent of the objects.

material([ka kd ks n sc]) sets the ambient/diffuse/specular strength, specular exponent, and specular color reflectance of the objects.

material default sets the ambient/diffuse/specular strength, specular exponent, and specular color reflectance of the objects to their defaults.

Remarks The material command sets the AmbientStrength, DiffuseStrength, SpecularStrength, SpecularExponent, and SpecularColorReflectance properties of all surface and patch objects in the axes. There must be visible light objects in the axes for lighting to be enabled. Look at the material.m M-file to see the actual values set (enter the command: type material).

Material	Ambient strength	Diffuse strength	Specular strength	Specular exponent	Specular color reflectance
Shiny	0.3	0.6	0.9	20	1.0
Dull	0.3	0.8	0.0	10	1.0
Metal	0.3	0.3	1.0	25	0.5

figure

```
[X, Y, Z]=sphere(50);
```

```
subplot(2,2,1)
```

```
axis equal
```

```
axis off
```

```
surface(X, Y, Z, 'FaceColor', [1 0 0], 'EdgeColor', 'none');
```

```
camlight;
```

```
lighting phong;
```

```
subplot(2,2,2)
```

```
axis equal
```

```
axis off
```

```
surface(X, Y, Z, 'FaceColor', [1 0.7 0], 'EdgeColor', 'none');
```

```
material shiny;
```

```
camlight;
```

```
lighting phong;
```

```
subplot(2,2,3)
```

```
axis equal
```

```
axis off
```

```
surface(X, Y, Z, 'FaceColor', [0 0.7 1], 'EdgeColor', 'none');
```

```
material dull;
```

```
camlight;
```

```
lighting phong;
```

```
subplot(2,2,4)
```

```
axis equal
```

```
axis off
```

```
surface(X, Y, Z, 'FaceColor', [0.95 0.95 0.95], 'EdgeColor', 'none');
```

```
material metal;  
camlight;  
lighting phong;
```

## 11.8. Прозрачность

### 11.9. Normals

Эксперименты с VertexNormals см. в ChessBoard

### 11.10. 3d графика. Примеры

```
%cube.m  
  
V = [  
0 0 0  
1 0 0  
0 1 0  
1 1 0  
0 0 1  
1 0 1  
0 1 1  
1 1 1];  
  
F = [  
2 4 3 1  
5 7 8 6  
8 7 3 4  
5 6 2 1  
1 3 7 5  
6 8 4 2];  
  
figure;  
axis equal;  
axis([-0.2 1.2 -0.2 1.2 -0.2 1.2]);  
view(3)  
camlight right;  
  
patch('Vertices', V, 'Faces', F, 'FaceVertexCData', [.5 .5 1], ...  
      'FaceColor', 'flat', 'BackFaceLighting', 'lit', ...
```



```
'EdgeColor', 'white', 'FaceLighting', 'phong')

ico

sph80

%ch3.m

n = 20;
X = rand(n, 3);
figure, hold on, grid on
plot3(X(:,1),X(:,2),X(:,3),'LineStyle','none','Marker','.', 'Ma
view(30,30);
pause;
stem3(X(:,1),X(:,2),X(:,3),'filled','-.-');
pause;


K = convhulln(X);
m = size(K,1);

patch('Vertices', X, 'Faces', K, 'FaceVertexCData', cool(m),...
      'FaceColor', 'flat', 'FaceAlpha', 0.7)
pause;

for k=30:400, view(k, 30), pause(0.05); end;

checking

human
nancy
```



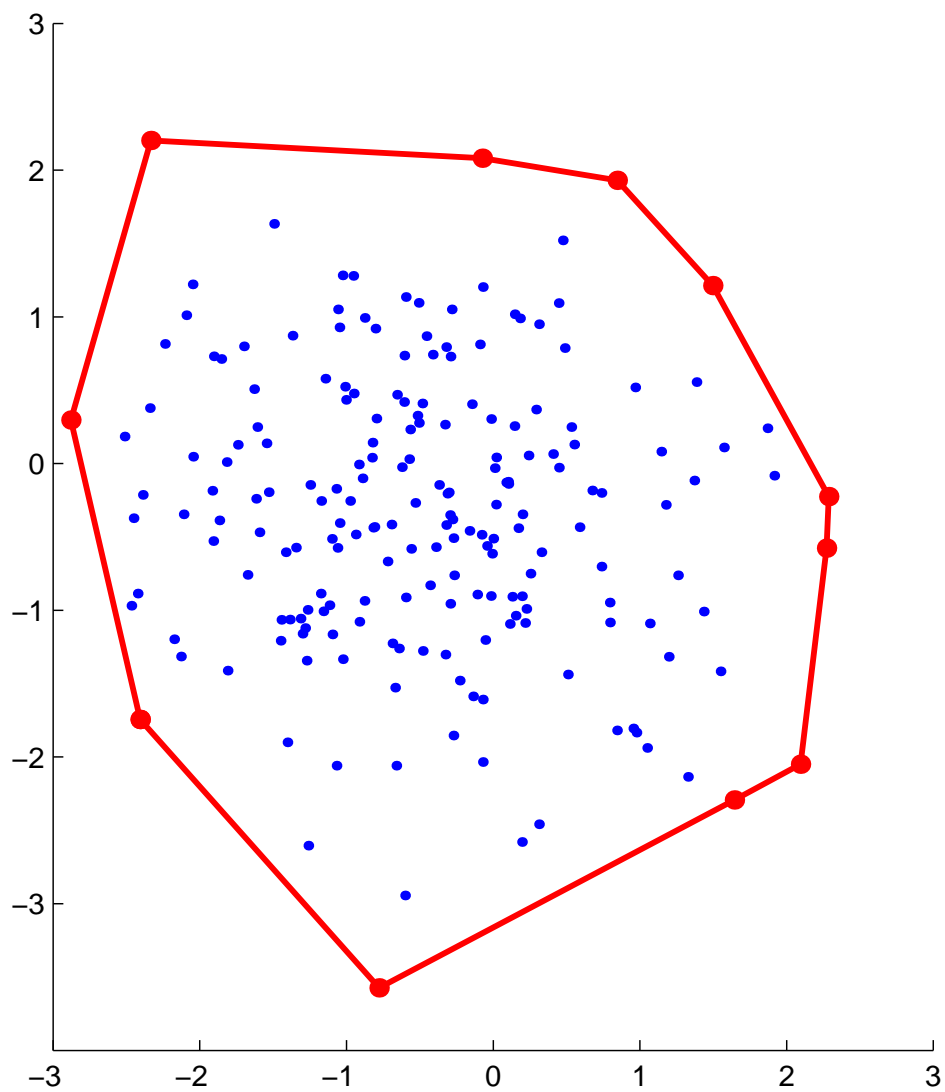


Рис. 11.2. Пример создания графического объекта

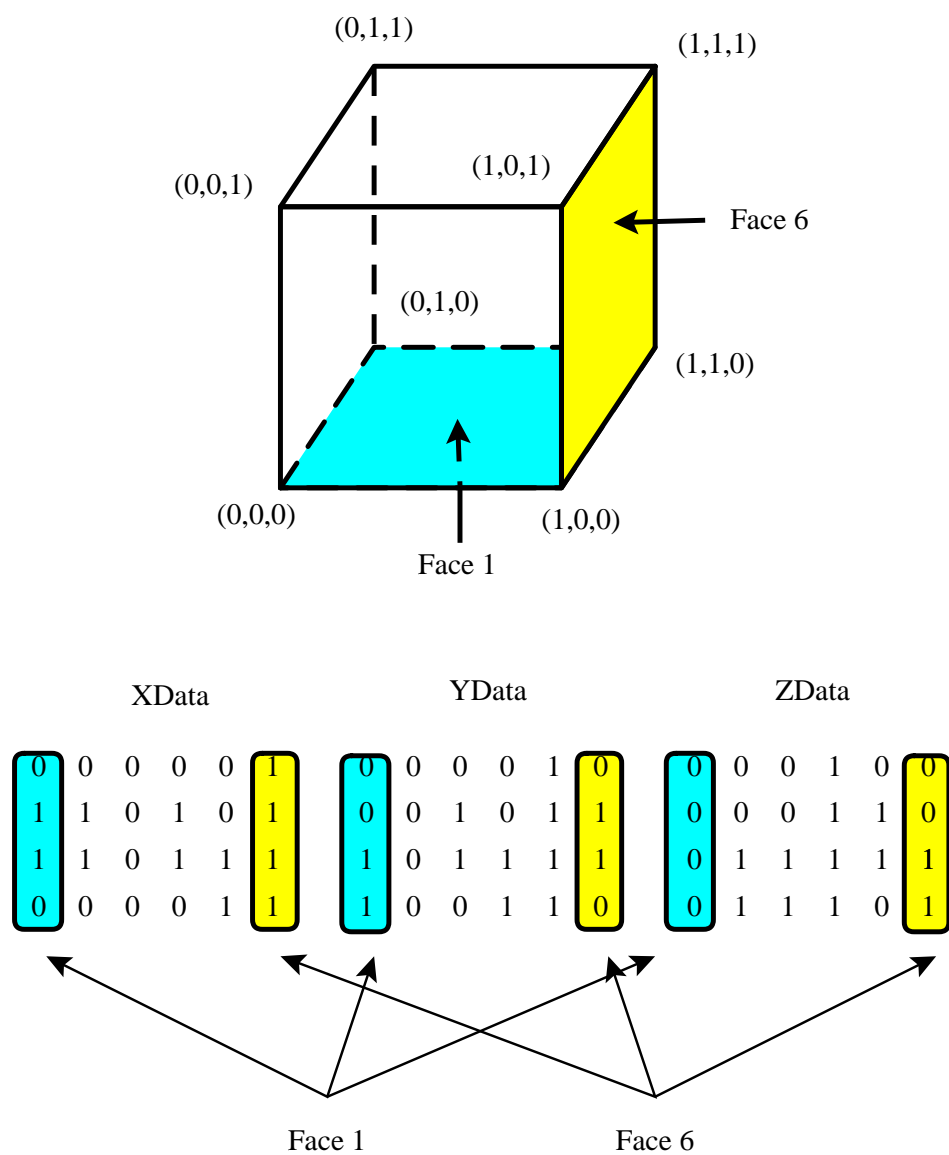


Рис. 11.3. Свойства XData, YData, ZData объекта Patch

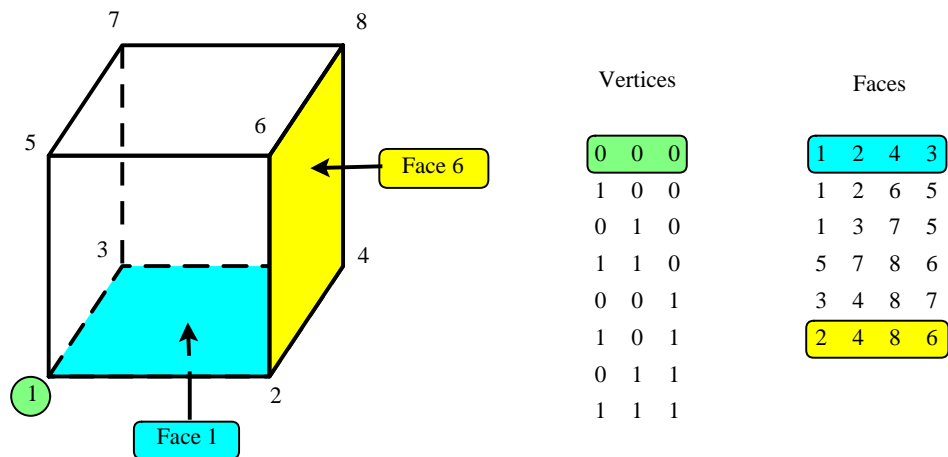


Рис. 11.4. Свойства Faces и Vertices объекта Patch

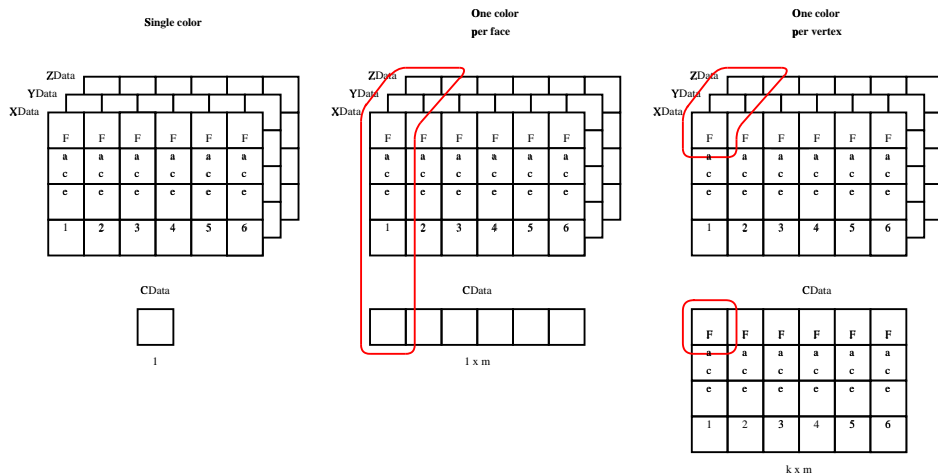


Рис. 11.5. Относительный способ задания цвета (indexed) объекта Patch, состоящего из 6 4-угольных граней. Грани заданы определением свойств XData, YData, ZData

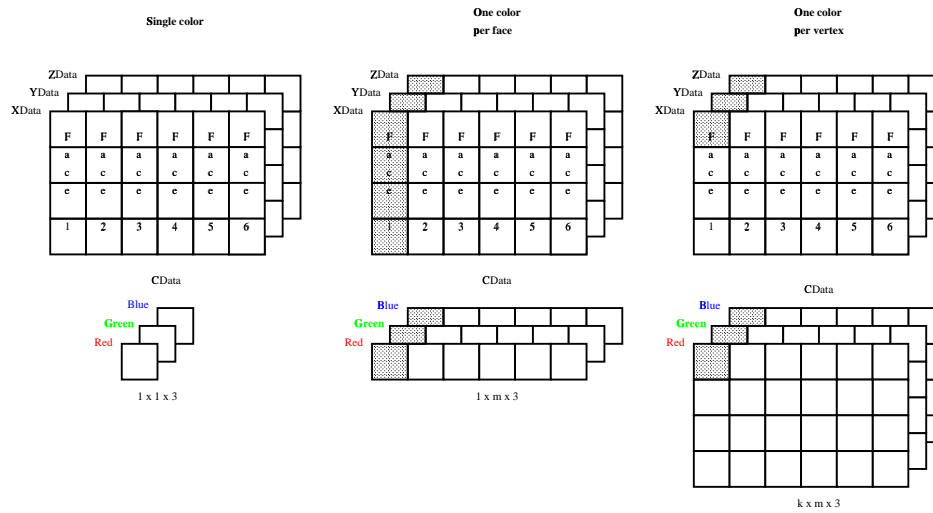


Рис. 11.6. Абсолютный способ задания цвета (true color) объекта Patch, состоящего из 6 4-угольных граней. Грани заданы определением свойств XData, YData, ZData

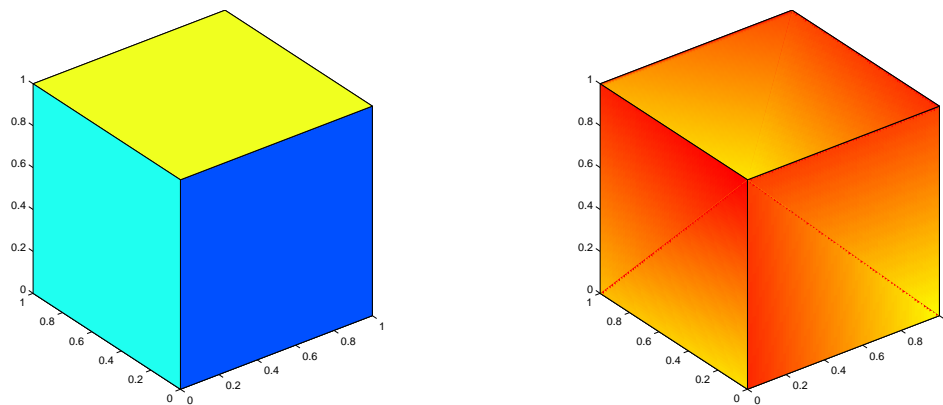


Рис. 11.7. Использование свойства CData

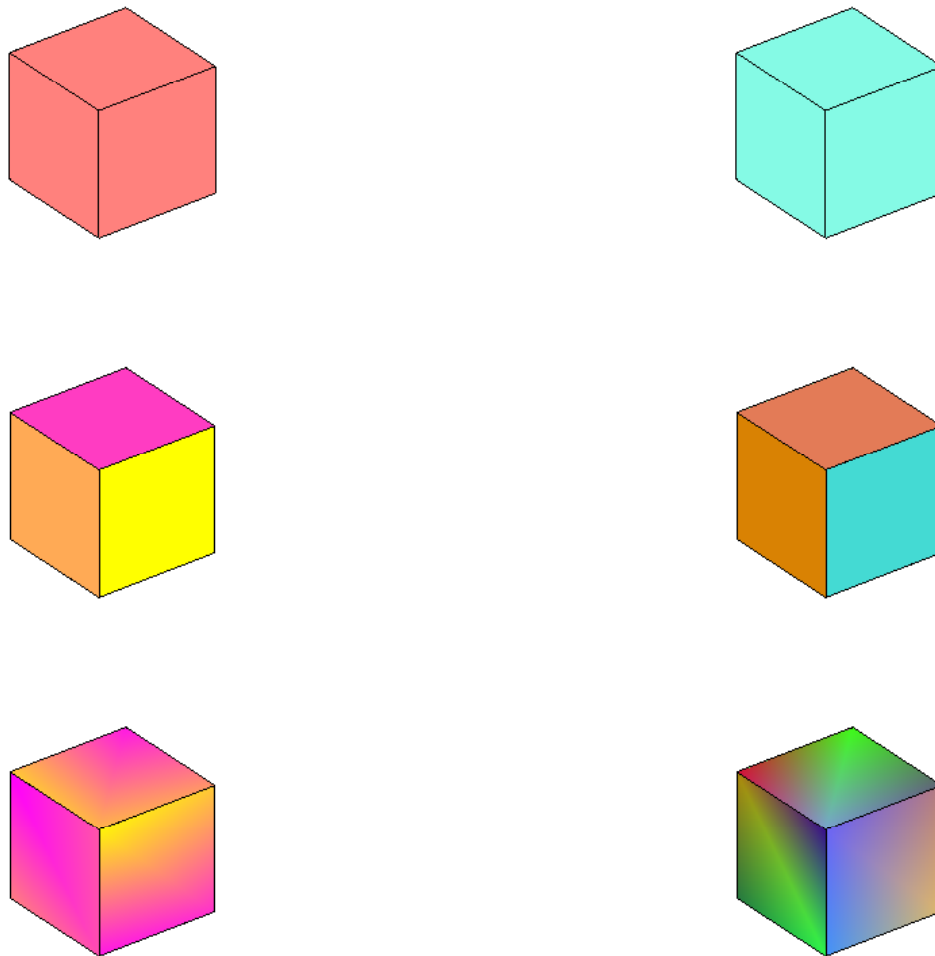


Рис. 11.8. Разные способы задания цвета объекта с помощью свойства CData

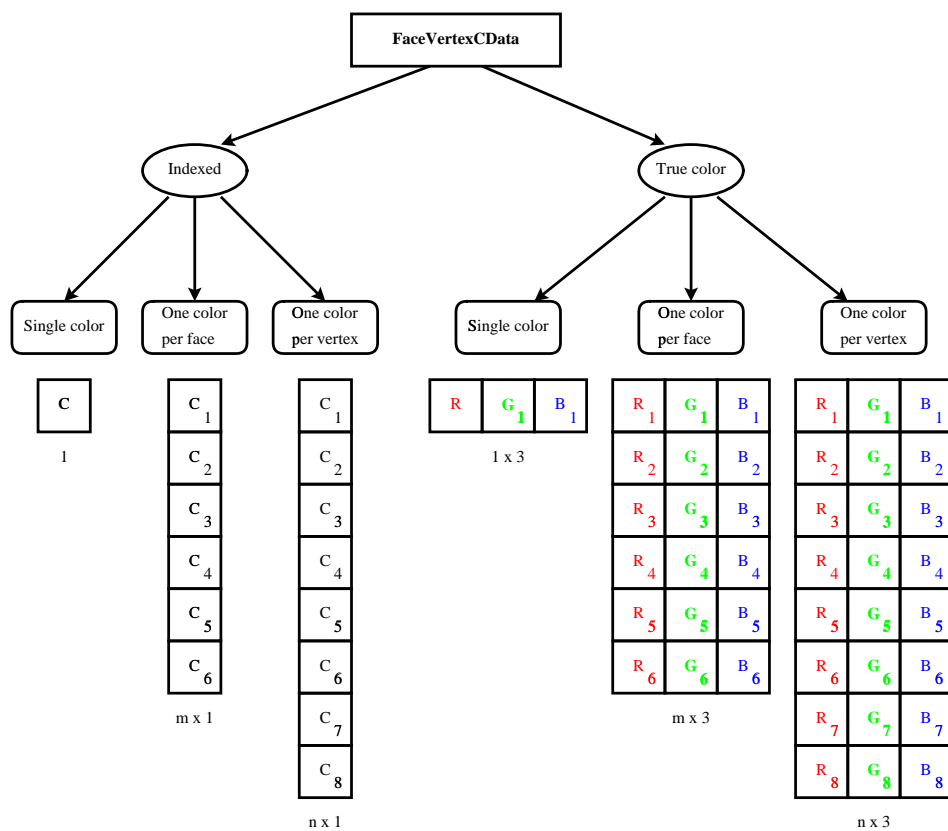


Рис. 11.9. Задание цвета объекта Patch с 8 вершинами и 6 гранями. Грани заданы определением свойств `Faces`, `Vertices`

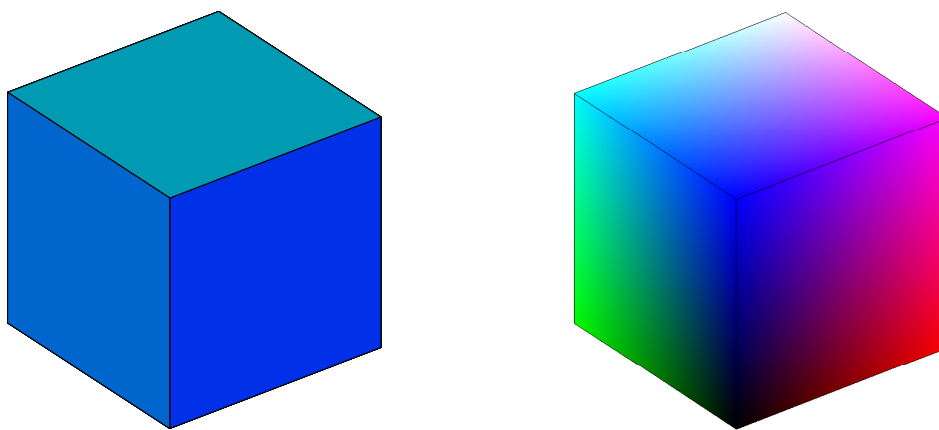


Рис. 11.10. Использование свойства `FaceVertexCData`

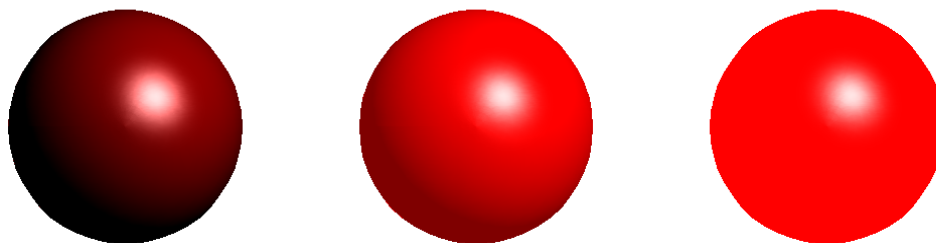


Рис. 11.11. AmbientStrength = 0, 0.5, 1

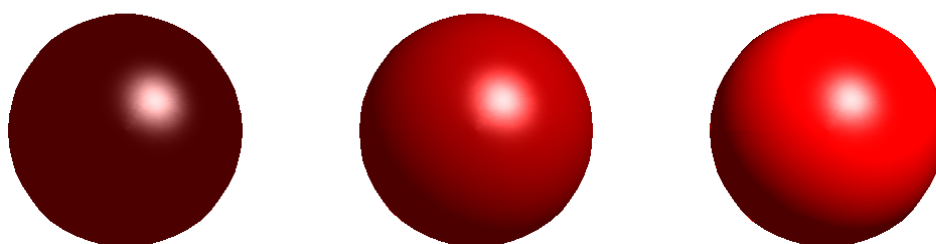


Рис. 11.12. DiffuseStrength = 0, 0.5, 1

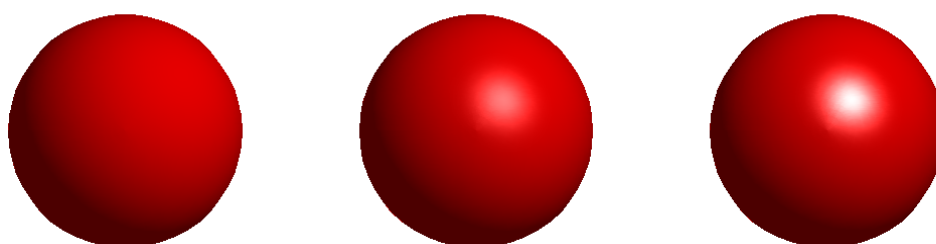


Рис. 11.13. SpecularStrength = 0, 0.5, 1

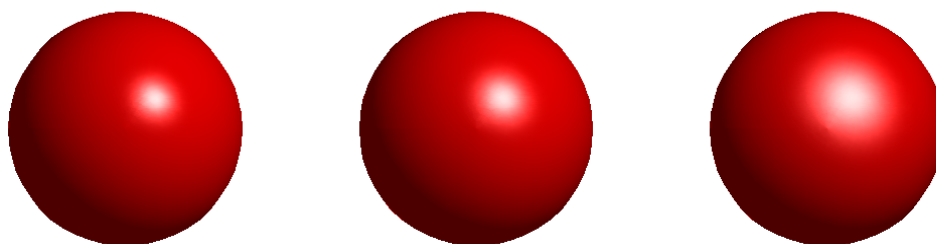


Рис. 11.14. SpecularExponent = 20, 13, 5



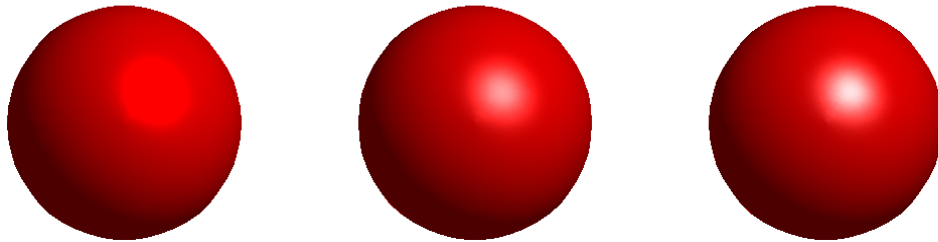


Рис. 11.15. SpecularColorReflectance = 0, 0.7, 1

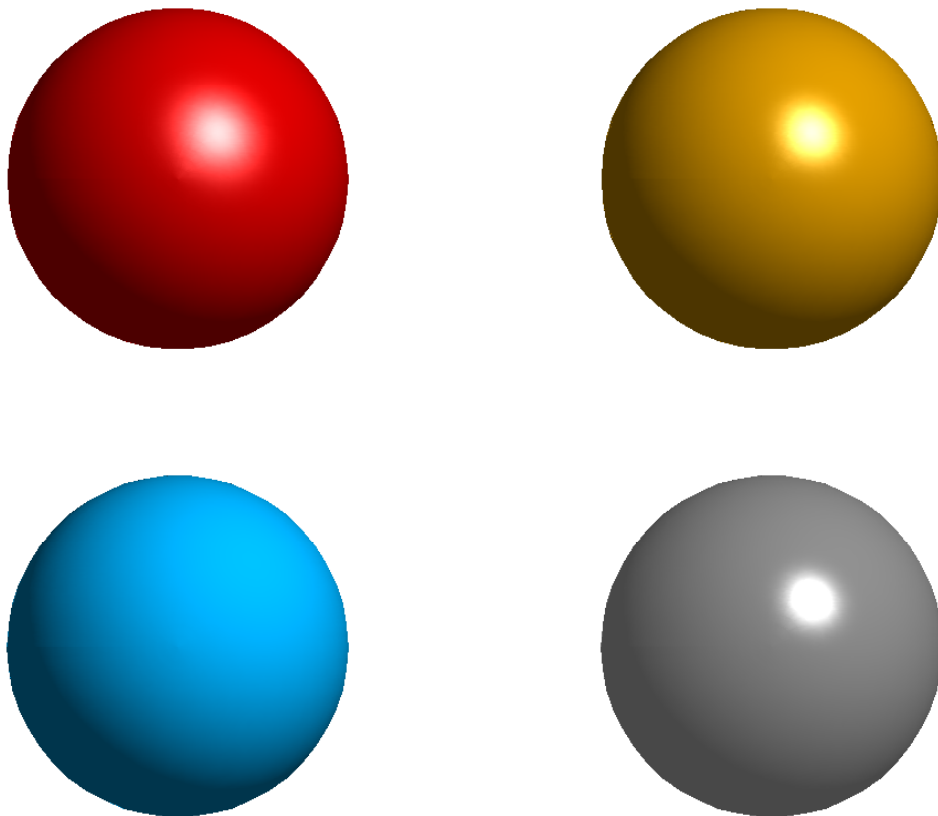


Рис. 11.16. Материал: default, shiny, dull, metal

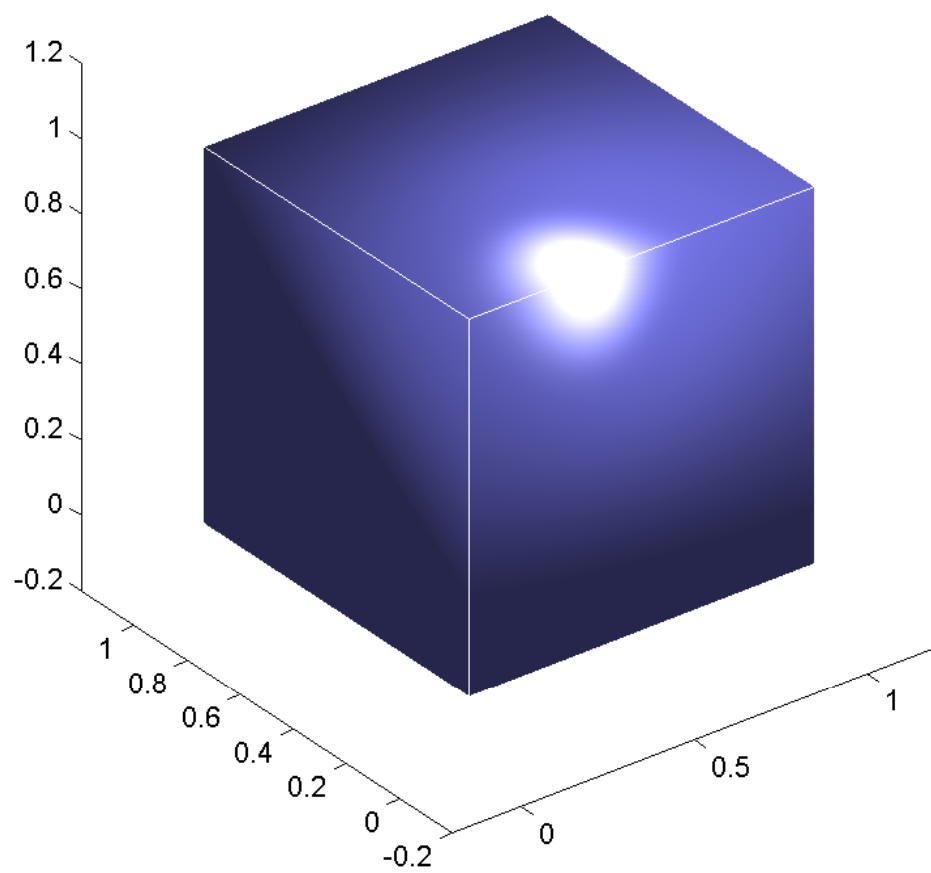


Рис. 11.17. Куб

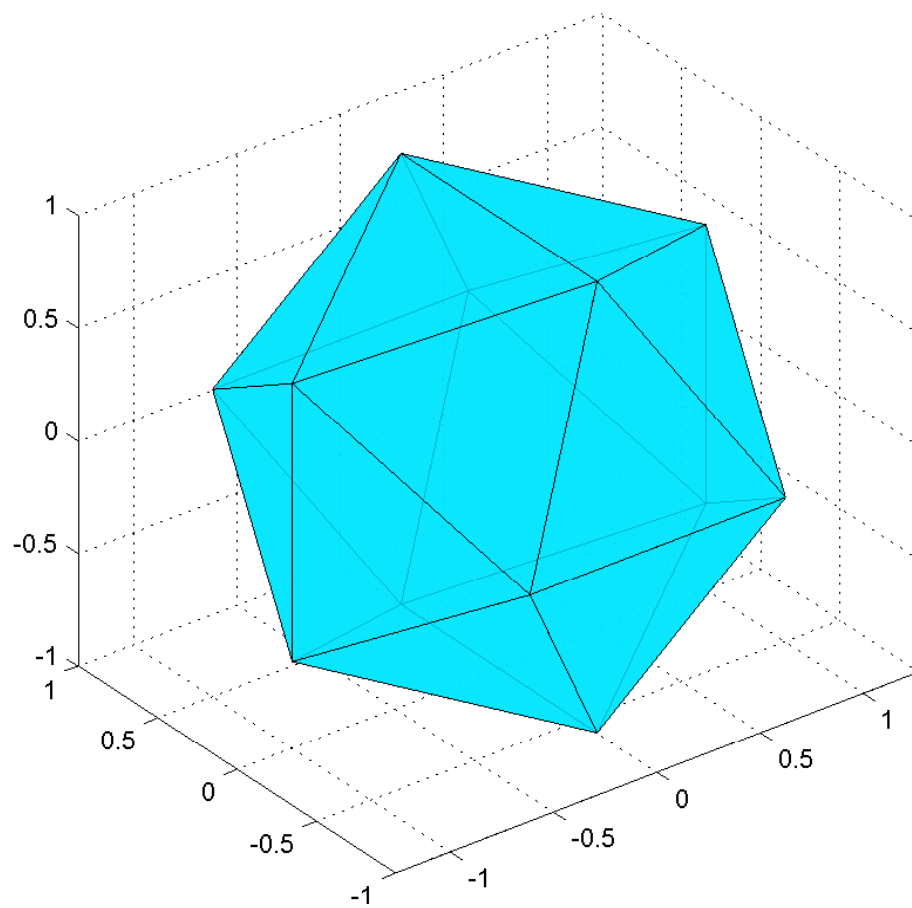


Рис. 11.18. Икосаэдр

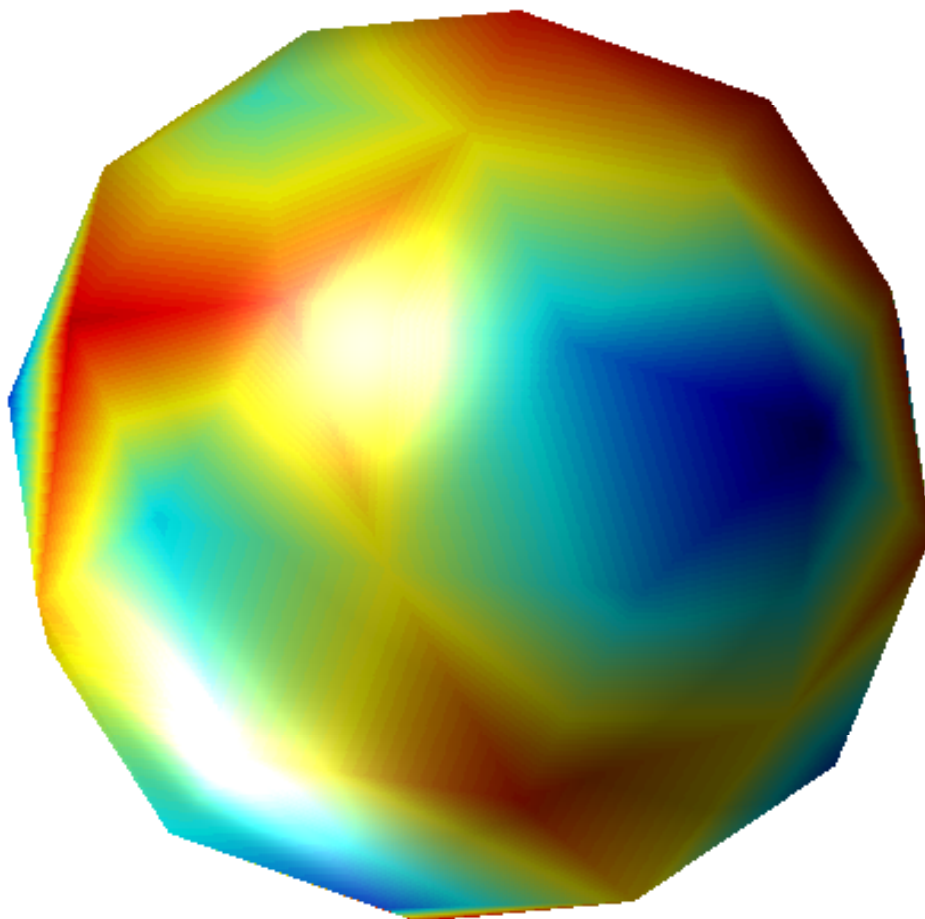


Рис. 11.19. Сфера из 80 треугольников

## Создание графического интерфейса пользователя

### 12.1. Элементы управления `uicontrol`

Система MATLAB предоставляет удобные средства для создания приложений, поддерживающих графический интерфейс пользователя (GUI). К стандартным элементам такого интерфейса относятся:

- Push buttons — кнопки,
- Toggle buttons — выключатели,
- Radio buttons — переключатели,
- Checkboxes — флажки,
- Edit text — строки ввода,
- Static text — статический текст,
- Sliders — полосы прокрутки,
- Frames — рамки для визуального объединения некоторых компонент,
- List boxes — списки,
- Popup menus — выпадающие меню.

## 12.2. Этапы разработки графического интерфейса

- Определение внешнего вида приложения: размещение элементов GUI
- Определение функциональности приложения: программирование элементов GUI

Как правило, при создании приложений с графическим интерфейсом пользователя разделяют описание внешнего вида и описание функциональности приложения. Если для создания GUI-приложения используется GUIDE, то

- внешний вид приложения описан в бинарном fig-файле,
- функциональность описана в m-файле.

С помощью GUIDE можно создать fig-файл и заготовку для m-файла.

## 12.3. Начало работы с GUIDE

Запустить GUIDE можно либо командой `guid`, либо с помощью меню `FILE|NEW|GUI`.

## 12.4. Параметры создаваемого приложения

В первую очередь необходимо задать параметры создаваемого приложения. С помощью меню `TOOLS|APPLICATION OPTIONS...` можно вызвать диалоговое окно, в котором задать следующие параметры:

- Параметры изменения размеров окна,
- Параметры, регулирующие взаимодействие с командной строкой,
- Характеристики автоматически создаваемого m-файла.

### 12.4.1. Параметры изменения размеров окна

Возможны 3 режима:

- `Non-resizable` — не возможно изменить размер окна,
- `Proportional` — MATLAB автоматически масштабирует размеры элементов окна,

- User-specified — при изменении размеров окна вызывается Callback-функция `ResizeFcn`, в которой программист сам должен задать изменение размеров элементов.

#### 12.4.2. Параметры, регулирующие взаимодействие с командной строкой

Когда MATLAB создает графическое изображение, дескрипторы объектов `figure` и `axes` включаются в список потомков их непосредственных предков и становятся доступны в командах `findobj`, `set`, `get` и других, подобных им. После следующей графической команды графический вывод направляется в текущее окно в зависимости от состояния поля `'NextPlot'` текущих объектов `figure` и `axes`.

Обычно в программах с графическим интерфейсом пользователя не желательно, чтобы дескрипторы графических элементов были доступны из командной строки и других программ. С другой стороны, часто в программах, содержащих объект `Axes` удобно управлять этим объектом из командной строки.

Можно сделать невидимыми дескрипторы любого окна и всех его потомков, управляя флагом `ACCESS OPTIONS` диалогового окна `APPLICATIONS OPTIONS`. Возможны 3 режима:

- Off — закрывает доступ к окну из командной строки (по умолчанию) (рекомендуется для программ с графическим интерфейсом пользователя).
- On — делает видимыми дескрипторы графических объектов из командного окна (рекомендуется для программ с графическим выводом).
- HandleVisibility — доступом к дескрипторам графических элементов окна управляет свойства `'HandleVisibility'` объекта `figure`.

Установка свойства `HandleVisibility` в состояние `'off'` (в GUIDE — по умолчанию) удаляет дескриптор окна из списка потомков объекта `root`. Таким образом, окно не может стать текущим. Однако прямой доступ к окну по дескриптору возможен.

Установка свойства `HandleVisibility` в состояние `'callback'` делает окно видимым из callback-функций и невидимым из командного окна.

Если значение поля `HandleVisibility` равно `'on'`, то все дескрипторы видны из командного окна.

## 12.5. Callback-функции

### 12.5.1. Callback-функции всех типов графических объектов

Все графические объекты имеют 4 callback-функции:

- **CallBack** — МАТЛАВ выполняет эту функцию всякий раз, когда пользователь нажимает на левую кнопку мыши и курсор мыши указывает на объект,
- **ButtonDownFcn** — МАТЛАВ выполняет эту функцию всякий раз, когда пользователь нажимает на левую кнопку мыши и курсор мыши указывает на объект (см. далее порядок выполнения callback-функций),
- **CreateFcn** — МАТЛАВ вызывает эту функцию сразу после создания объекта.
- **DeleteFcn** — МАТЛАВ вызывает эту функцию непосредственно перед удалением объекта.

### 12.5.2. Callback-функции объекта Figure

Дополнительные callback-функции объекта Figure:

- **CloseRequestFcn** — МАТЛАВ выполняет эту функцию при получении команды закрыть окно.
- **KeyPressFcn** — МАТЛАВ выполняет эту функцию, если пользователь нажимает на клавиатурную клавишу и курсор находится внутри окна.
- **ResizeFcn** — МАТЛАВ выполняет эту функцию, если пользователь изменяет размеры окна.
- **WindowButtonDownFcn** — МАТЛАВ выполняет эту функцию, если пользователь нажимает на левую кнопку мыши и курсор находится внутри окна, но не на одном из элементов управления.
- **WindowButtonMotionFcn** — МАТЛАВ выполняет эту функцию, когда пользователь перемещает курсор мыши внутри окна.
- **WindowButtonUpFcn** — МАТЛАВ выполняет эту функцию, когда пользователь отпускает левую кнопку мыши после того, как кнопка была нажата и в момент нажатия курсор находился внутри окна.



### 12.5.3. Порядок вызова callback-функций

Элемент управления в активном состоянии после нажатия левой кнопки мыши выполняет функцию **Callback**. Если элемент находится в неактивном состоянии, то вначале выполняется функция **он**, а затем **ButtonDownFcn** элемента управления.

Установка свойства элемента управления **enable** в **on** переводит элемент в активное состояние. Установка свойства **enable** в **off** или в **inactive** переводит элемент в неактивное состояние. Отличие **off** от **inactive** в том, что в первом случае надпись на элементе окрашивается в серый цвет, а во втором элемент выглядит как обычный.

## 12.6. Функция *guihandles*

Функция

```
handles = guihandles(object_handle)
```

создает структуру, содержащую дескрипторы всех графических объектов (включая элементы **uicontrol**, объекты **axes** и их потомков), расположенных в графическом окне. Дескриптор **object\_handle** должен указывать на это окно, либо на любого из его потомков. Именами полей в создаваемой структуре являются метки **Tag** элементов. Если поле **Tag** элемента пусто, то поле для этого элемента не создается. Если несколько элементов имеют одинаковый **Tag**, то соответствующее поле в структуре является вектором. При вызове функции **guihandles** без параметров:

```
handles = guihandles
```

создается структура дескрипторов для текущего окна.

Программа **GUIDE** всегда добавляет в начало создаваемого ею m-файла код, сохраняющий структуру дескрипторов элементов в данных графического приложения.

## 12.7. Функция *guidata*

Функция **guidata** предназначена для того, чтобы сохранять и читать данные графического приложения.

Возможные варианты вызова функции:

```
guidata(object_handle,data)  
data = guidata(object_handle)
```

Функция `guidata(object_handle,data)` сохраняет данные `data` как данные графического приложения. Параметр `object_handle` должен быть либо дескриптором окна, либо любого его потомка.

Функция `data=guidata(object_handle)` возвращает сохраненные данные графического приложения или возвращает пустую матрицу, если данных графического приложения нет.

Программа GUIDE всегда добавляет в начало создаваемого ею m-файла код, сохраняющий структуру `guihandles` в данных графического приложения:

```
fig = openfig(mfilename,'reuse');
handles = guihandles(fig);
guidata(fig, handles);
```

Examples In this example, `guidata` is used to save a structure on a GUI figure's application data from within the initialization section of the application M-file. This structure is initially created by `guihandles` and then used to save additional data as well.

```
data = guihandles(ffigure_handle); % create structure of handles
data.numberOfErrors = 0; % add some additional data
guidata(ffigure_handle,data) % save the structure
```

You can recall the data from within a subfunction callback routine and then save the structure again:

```
data = guidata(gcbo); % get the structure in the subfunction
data.numberOfErrors = data.numberOfErrors + 1;
guidata(gcbo,data) % save the changes to the structure
```

## 12.8. Примеры

**Пример 25.** Покажем, как можно запрограммировать группу переключателей, чтобы во всей группе можно было отметить лишь один переключатель. 2 переключателя:

```
function varargout = Radio2(varargin)
% RADIO Application M-file for Radio.fig
%   FIG = RADIO launch Radio GUI.
%   RADIO('callback_name', ...) invoke the named callback.
```

```
% Last Modified by GUIDE v2.0 13-May-2002 20:18:49
```

```

if nargin == 0 % LAUNCH GUI

fig = openfig(mfilename,'reuse');

% Use system color scheme for figure:
set(fig,'Color',get(0,'defaultUicontrolBackgroundColor'));

% Generate a structure of handles to pass to callbacks, and store it.
handles = guihandles(fig);
guidata(fig, handles);

if narginout > 0
varargout{1} = fig;
end

elseif ischar(varargin{1}) % INVOKE NAMED SUBFUNCTION OR CALLBACK

try
[varargout{1:nargout}] = feval(varargin{:}); % FEVAL switchyard
catch
disp(lasterr);
end

end

% -----
function varargout = radiobutton1_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol h=handles.radiobutton1.
set(handles.radiobutton2,'Value',1-get(h,'Value'));

% -----
function varargout = radiobutton2_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol h=handles.radiobutton2.
set(handles.radiobutton1,'Value',1-get(h,'Value'));

```

**Пример 26.** 3 переключателя:

```
function varargout = Radio3(varargin)
```

```

% RADIO Application M-file for Radio.fig
%   FIG = RADIO launch Radio GUI.
%   RADIO('callback_name', ...) invoke the named callback.

% Last Modified by GUIDE v2.0 13-May-2002 20:18:49

if nargin == 0 % LAUNCH GUI

fig = openfig(mfilename,'reuse');

% Use system color scheme for figure:
set(fig,'Color',get(0,'defaultUicontrolBackgroundColor'));

% Generate a structure of handles to pass to callbacks, and store it.
handles = guihandles(fig);
guidata(fig, handles);

if nargout > 0
varargout{1} = fig;
end

elseif ischar(varargin{1}) % INVOKE NAMED SUBFUNCTION OR CALLBACK

try
[varargout{1:nargout}] = feval(varargin{:}); % FEVAL switchyard
catch
disp(lasterr);
end

end

% -----
function varargout = radiobutton1_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.radiobutton1.
if get(h,'Value')==0
    set(h,'Value',1);
else
    set(handles.radiobutton2,'Value',0);
    set(handles.radiobutton3,'Value',0);

```



Рис. 12.1. Переключатели

```

end;

% -----
function varargout = radiobutton2_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.radiobutton2.
if get(h,'Value')==0
    set(h,'Value',1);
else
    set(handles.radiobutton1,'Value',0);
    set(handles.radiobutton3,'Value',0);
end;

% -----
function varargout = radiobutton3_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.radiobutton3.
if get(h,'Value')==0
    set(h,'Value',1);
else
    set(handles.radiobutton1,'Value',0);
    set(handles.radiobutton2,'Value',0);
end;

```

**Пример 27.** Приведем пример программы, работающей с двумя спис-

ками файлов:

```
function varargout = lpgo(varargin)
% LPGO Application M-file for lpgo.fig
%   FIG = LPGO launch lpgo GUI.
%   LPGO('callback_name', ...) invoke the named callback.

% Last Modified by GUIDE v2.0 14-May-2002 12:26:57

if nargin == 0 % LAUNCH GUI

fig = openfig(mfilename,'reuse');

% Use system color scheme for figure:
set(fig,'Color',get(0,'defaultUicontrolBackgroundColor'));

% Generate a structure of handles to pass to callbacks, and store it.
handles = guihandles(fig);
guidata(fig, handles);
% Populate the lists
load_list(handles);

if nargout > 0
varargout{1} = fig;
end

elseif ischar(varargin{1}) % INVOKE NAMED SUBFUNCTION OR CALLBACK

try
[varargout{1:nargout}] = feval(varargin{:}); % FEVAL switchyard
catch
disp(lasterr);
end

end

% -----
function varargout = Input_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.Input.
```

```

names=get(h,'String');
ind=get(h,'Value');
set(handles.Text,'String',names{ind});

% -----
function varargout = Output_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.Output.
names=get(h,'String');
ind=get(h,'Value');
set(handles.Edit,'String',names{ind});

% -----
function varargout = Go_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.Go.
disp('pushbutton1 Callback not implemented yet.')

% -----
function varargout = Close_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.Close.
delete(handles.figure1);

% -----
% Read the current directory and sort the names
% -----
function load_list(handles)
dir_struct = dir(pwd)
names = sortrows({dir_struct.name}');
set(handles.Input,'String',names,'Value',1);
set(handles.Output,'String',names,'Value',1);
set(handles.Edit,'String',names{1});

```

**Пример 28.** Программа построения интерполянтов:

```

function varargout = interpolation(varargin)
% INTERPOLATION Application M-file for interpolation.fig
% FIG = INTERPOLATION launch interpolation GUI.

```

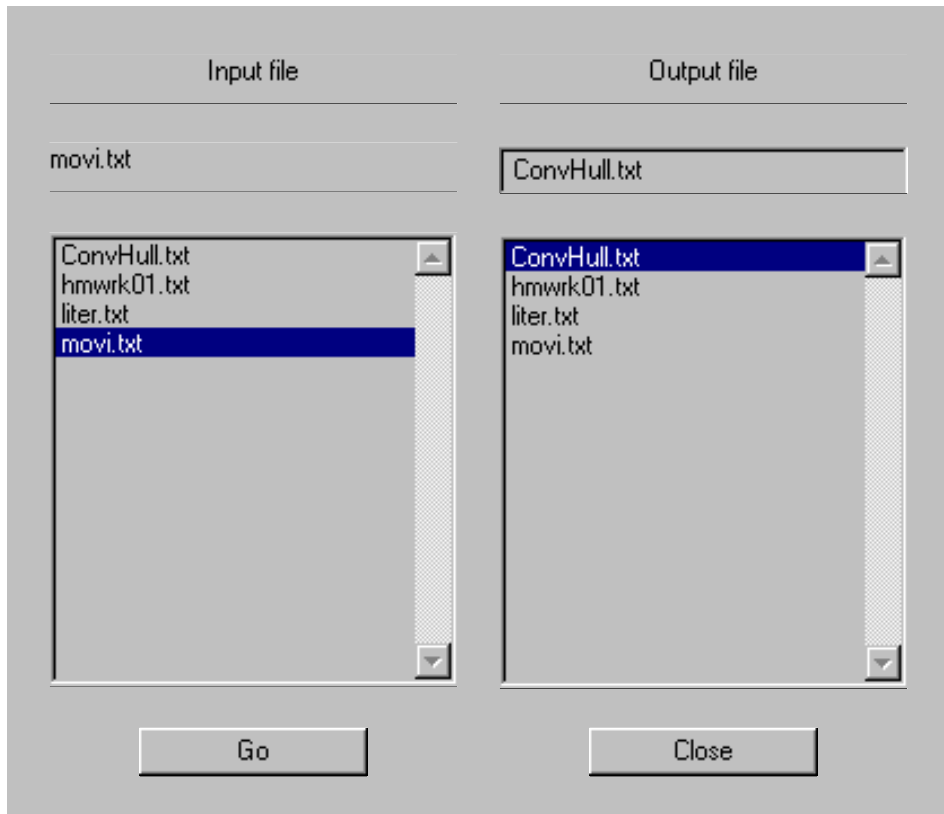


Рис. 12.2. Пример программы, работающей с двумя списками файлов

```
% INTERPOLATION('callback_name', ...) invoke the named callback.

% Last Modified by GUIDE v2.0 13-May-2002 19:52:54

if nargin == 0 % LAUNCH GUI

    fig = openfig(mfilename,'reuse');

    % Generate a structure of handles to pass to callbacks, and store it.
    handles = guihandles(fig);
    guidata(fig, handles);

    if nargout > 0
        varargout{1} = fig;
    end
end
```



```
elseif ischar(varargin{1}) % INVOKE NAMED SUBFUNCTION OR CALLBACK
```

```
    try
        [varargout{1:nargout}] = feval(varargin{:}); % FEVAL switchyard
    catch
        disp(lasterr);
    end
end
```

```
end
```

```
% -----
function Refresh(handles)
RefreshData(handles);
RefreshLinear(handles);
RefreshCubic(handles);
RefreshSpline(handles);
```

```
% -----
function RefreshData(handles)
delete(findobj(handles.XY,'Tag','Data'));
if get(handles.Plot,'Value')==1
    x=0;
    y=0;
    x=eval(get(handles.X,'String'));
    y=eval(get(handles.Y,'String'));
    line('XData',x,'YData',y,'Marker','.', 'LineStyle','none',...
        'MarkerSize',15,'Color','b','Parent',handles.XY,'Tag','Data');
end;
```

```
% -----
function RefreshLinear(handles)
delete(findobj(handles.XY,'Tag','Linear'));
if get(handles.Linear,'Value')==1
    x=0;
    y=0;
    x=eval(get(handles.X,'String'));
    y=eval(get(handles.Y,'String'));
    line('XData',x,'YData',y,'Marker','.', 'LineStyle','none',...
        'MarkerSize',15,'Color','b','Parent',handles.XY,'Tag','Linear');
```

```

        y=eval(get(handles.Y,'String'));
        line('XData',x,'YData',y,'Color','k','LineWidth',2,...
            'Parent',handles.XY,'Tag','Linear');
    end;

% -----
function RefreshCubic(handles)
delete(findobj(handles.XY,'Tag','Cubic'));
if get(handles.Cubic,'Value')==1
    x=0;
    y=0;
    x=eval(get(handles.X,'String'));
    y=eval(get(handles.Y,'String'));
    xx=min(x):0.01:max(x);
    yy=pchip(x,y,xx);
    line('XData',xx,'YData',yy,'Color','r','LineWidth',2,...
        'Parent',handles.XY,'Tag','Cubic');
end;

% -----
function RefreshSpline(handles)
delete(findobj(handles.XY,'Tag','Spline'));
if get(handles.Spline,'Value')==1
    x=0;
    y=0;
    x=eval(get(handles.X,'String'));
    y=eval(get(handles.Y,'String'));
    xx=min(x):0.01:max(x);
    yy=spline(x,y,xx);
    line('XData',xx,'YData',yy,'Color','m','LineWidth',2,...
        'Parent',handles.XY,'Tag','Spline');
end;

% -----
function varargout = Plot_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.Plot.
RefreshData(handles);

```

```
% -----  
function varargout = Linear_Callback(h, eventdata, handles, varargin)  
% Stub for Callback of the uicontrol handles.Linear.  
RefreshLinear(handles);  
  
% -----  
function varargout = Cubic_Callback(h, eventdata, handles, varargin)  
% Stub for Callback of the uicontrol handles.Cubic.  
RefreshCubic(handles);  
  
% -----  
function varargout = Spline_Callback(h, eventdata, handles, varargin)  
% Stub for Callback of the uicontrol handles.Spline.  
RefreshSpline(handles);  
  
% -----  
function varargout = X_Callback(h, eventdata, handles, varargin)  
% Stub for Callback of the uicontrol handles.X.  
Refresh(handles);  
  
% -----  
function varargout = Y_Callback(h, eventdata, handles, varargin)  
% Stub for Callback of the uicontrol handles.Y.  
Refresh(handles);  
  
% -----  
function varargout = Refresh_Callback(h, eventdata, handles, varargin)  
% Stub for Callback of the uicontrol handles.Refresh.  
Refresh(handles);  
  
% -----  
function varargout = Close_Callback(h, eventdata, handles, varargin)  
% Stub for Callback of the uicontrol handles.Close.  
delete(handles.figure1);
```

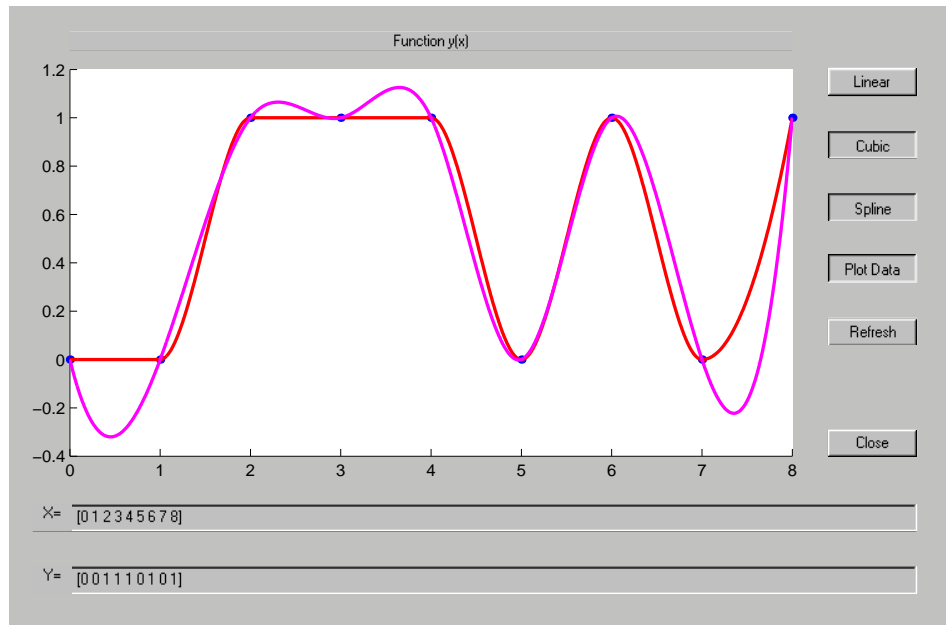


Рис. 12.3. Программа построения интерполянтов

**Пример 29.** Приведем текст программы, рисующей кривые второго порядка. Кривые задаются уравнением

$$a_{11}x^2 + 2a_{12}xy + a_{22}y^2 + 2a_1x + 2a_2y + \alpha = 0$$

или в матричной форме:

$$r^T A r + 2a^T r + \alpha = 0,$$

где

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, \quad a = \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}, \quad r = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}.$$

```
function varargout = quadrix(varargin)
% QUADRIX Application M-file for quadrix.fig
% FIG = QUADRIX launch quadrix GUI.
% QUADRIX('callback_name', ...) invoke the named callback.
```

```
% Last Modified by GUIDE v2.0 15-May-2002 16:56:32
```

```

if nargin == 0 % LAUNCH GUI

    fig = openfig(mfilename,'reuse');

    % Use system color scheme for figure:
    set(fig,'Color',get(0,'defaultUicontrolBackgroundColor'));

    % Generate a structure of handles to pass to callbacks, and store it.
    handles = guihandles(fig);
    guidata(fig, handles);
    redraw(handles);

    if nargout > 0
        varargout{1} = fig;
    end

elseif ischar(varargin{1}) % INVOKE NAMED SUBFUNCTION OR CALLBACK

    try
        [varargout{1:nargout}] = feval(varargin{:}); % FEVAL switchyard
    catch
        disp(lasterr);
    end

end

% -----
function varargout = a11_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.a11.
redraw(handles);

% -----
function varargout = a12_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.a12.
set(handles.a21,'String',get(h,'String'));
redraw(handles);

% -----

```

```
function varargout = a21_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.a21.
set(handles.a12,'String',get(h,'String'));
redraw(handles);
```

```
% -----
function varargout = a22_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.a22.
redraw(handles);
```

```
% -----
function varargout = a1_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.a1.
redraw(handles);
```

```
% -----
function varargout = a2_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.a2.
redraw(handles);
```

```
% -----
function varargout = alpha_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.alpha.
redraw(handles);
```

```
% -----
function varargout = xmin_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.xmin.
redraw(handles);
```

```
% -----
function varargout = xmax_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.xmax.
redraw(handles);
```

```

% -----
function varargout = ymin_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.ymin.
redraw(handles);

% -----
function varargout = ymax_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.ymax.
redraw(handles);

% -----
function varargout = Close_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.Close.
delete(handles.figure1);

% -----
function varargout = Redraw_Callback(h, eventdata, handles, varargin)
% Stub for Callback of the uicontrol handles.Redraw.
redraw(handles);

% -----
function redraw(handles)
a11=eval(get(handles.a11,'String'));
a12=eval(get(handles.a12,'String'));
a22=eval(get(handles.a22,'String'));
a1=eval(get(handles.a1,'String'));
a2=eval(get(handles.a2,'String'));
alpha=eval(get(handles.alpha,'String'));
xmin=eval(get(handles.xmin,'String'));
xmax=eval(get(handles.xmax,'String'));
ymin=eval(get(handles.ymin,'String'));
ymax=eval(get(handles.ymax,'String'));
syms x y
f=a11*x^2+2*a12*x*y+a22*y^2+2*a1*x+2*a2*y+alpha;
ezplot(f,[xmin xmax ymin ymax],handles.figure1);
set(handles.XY,'XGRid','on','YGrid','on');

```

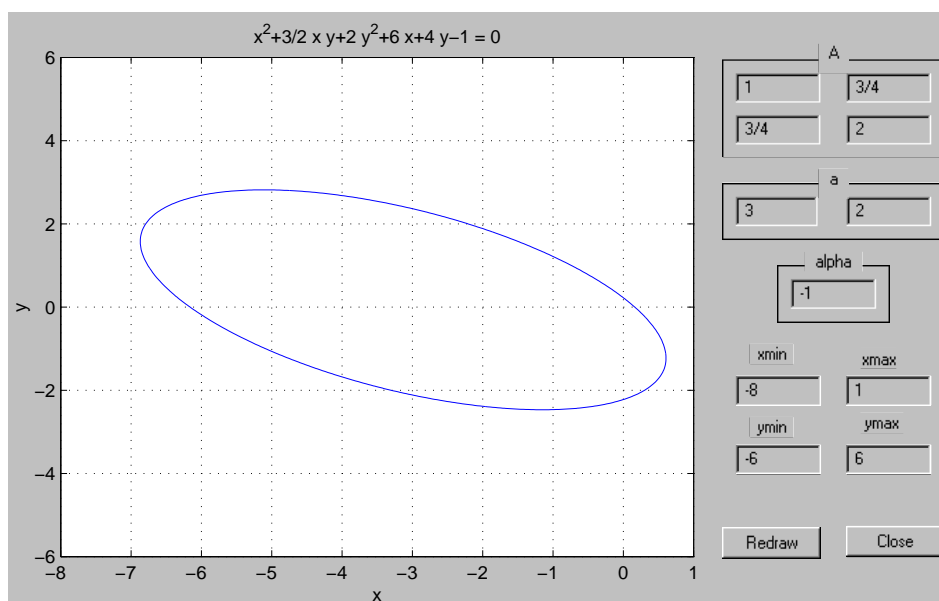


Рис. 12.4. Программа для изображения кривых второго порядка



## Литература

1. Потемкин В.Г. Система инженерных и научных расчетов МАТЛАВ 5.x: – В 2-х т. Том 1. – М.: ДИАЛОГ-МИФИ, 1999. – 304 с.
2. Потемкин В.Г. Система инженерных и научных расчетов МАТЛАВ 5.x: – В 2-х т. Том 2. – М.: ДИАЛОГ-МИФИ, 1999. – 366 с.
3. Потемкин В.Г., Рудаков П.И. МАТЛАВ 5.2. Студенческая версия. – М.: ДИАЛОГ-МИФИ, 1999. – 366 с.
4. Чен К., Джиблин П., Ирвинг А. МАТЛАВ в математических исследованиях. – М.: Мир, 2001. – 346 с.