Урок 6

Программные средства обычной графики

Одно из достоинств системы MATLAB – обилие средств графики, начиная от команд построения простых графиков функций одной переменной в декартовой системе координат и заканчивая комбинированными и презентационными графиками с элементами анимации, а также средствами проектирования графического пользовательского интерфейса (GUI). Особое внимание в системе уделено трехмерной графике с функциональной окраской отображаемых фигур и имитацией различных световых эффектов. Этот урок посвящен описанию основных программных средств обычной графики системы MATLAB [13, 16, 46]. Показано построение графиков как командами, исполняемыми из командной строки, так и с помощью простых программ с линейной структурой. Математические основы машинной графики описаны в [66] и здесь не рассматриваются. Материал этого урока относится к любой реализации базовой системы MATLAB.

6.1. Графики функций и данных

6.1.1. Построение графиков отрезками прямых

Функции одной переменной $y(x)$ находят широкое применение в практике математических и других расчетов, а также в технике компьютерного математического моделирования. Для отображения таких функций используются *графики* в декартовой (прямоугольной) системе координат. При этом обычно строятся две оси – горизонтальная X и вертикальная Y, и задаются координаты x и y, определяющие узловые точки функции $y(x)$. Эти точки соединяются друг с другом отрезками прямых, то есть при построении графика осуществляется *линейная интерполяция* для промежуточных точек. Поскольку MATLAB – матричная система, совокупность точек $y(x)$ задается векторами `X` и `Y` одинакового размера.

Команда `plot` служит для построения графиков функций в декартовой системе координат. Эта команда имеет ряд параметров, рассматриваемых ниже.

- `plot(X,Y)` строит график функции $y(x)$, координаты точек (x,y) которой берутся из векторов одинакового размера `Y` и `X`. Если `X` или `Y` – матрица, то строится семейство графиков по данным, содержащимся в колонках матрицы.

  Приведенный ниже пример иллюстрирует построение графиков двух функций – $\sin(x)$ и $\cos(x)$, значения функции которых содержатся в матрице `Y`, а значения аргумента x хранятся в векторе `X`:

  ```
  >> x=[0 1 2 3 4 5]; Y=[sin(x);cos(x)]; plot(x,Y)
  ```

  На рис. 6.1 показан график функций из этого примера. В данном случае отчетливо видно, что график состоит из отрезков, и если вам нужно, чтобы отображаемая функция имела вид гладкой кривой, необходимо увеличить количество узловых точек. Расположение их ординат может быть произвольным.

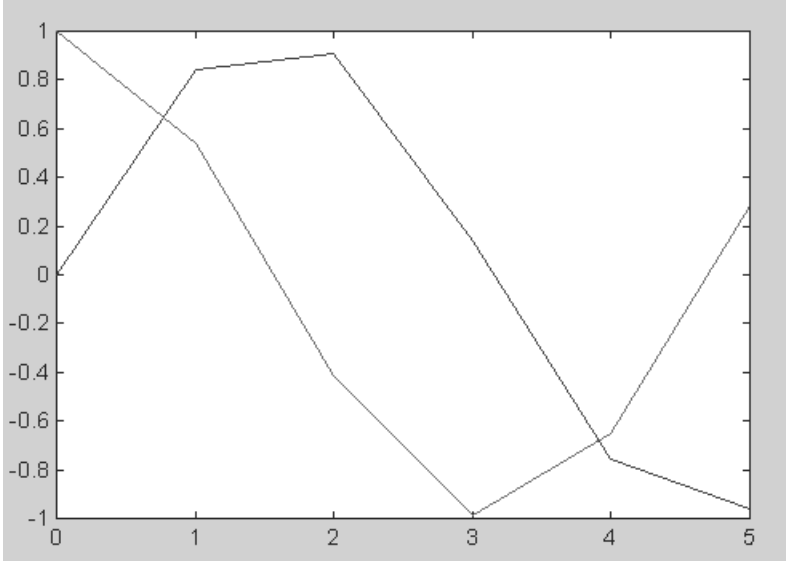

Рис. 6.1. Графики двух функций в декартовой системе координат

- `plot(Y)` строит график $y(i)$, где значения y берутся из вектора `Y`, а i представляет собой индекс соответствующего элемента. Если `Y` содержит комплексные элементы, то выполняется команда `plot(real(Y), imag(Y))`. Во всех других случаях мнимая часть данных игнорируется.
  Вот пример использования команды `plot(Y)`:

  ```
  >> x=-2*pi:0.02*pi:2*pi; y=sin(x)+i*cos(3*x); plot(y)
  ```

  Соответствующий график показан на рис. 6.2.
- `plot(X,Y,S)` аналогична команде `plot(X,Y)`, но тип линии графика можно задавать с помощью строковой константы `S`.
  Значениями константы `S` могут быть следующие символы.

Цвет линии	
`y`	Желтый
`m`	Фиолетовый
`c`	Голубой
`r`	Красный
`g`	Зеленый
`b`	Синий
`w`	Белый
`k`	Черный
Тип точки	
`.`	Точка
`o`	Окружность

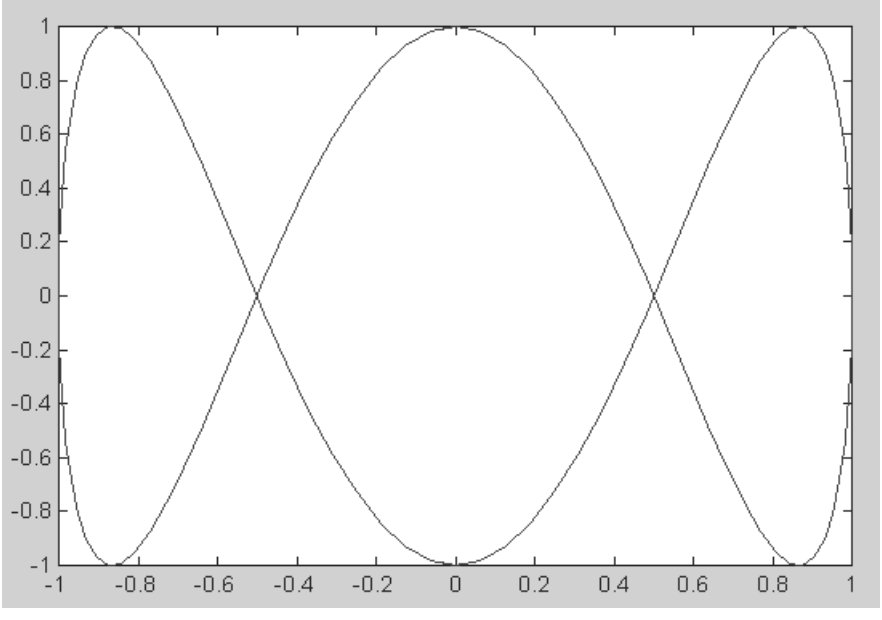

Рис. 6.2. График функции, представляющей вектор Y с комплексными элементами

Тип точки	
`x`	Крест
`+`	Плюс
`*`	Звездочка
`s`	Квадрат
`d`	Ромб
`v`	Треугольник (вниз)
`^`	Треугольник (вверх)
`<`	Треугольник (влево)
`>`	Треугольник (вправо)
`p`	Пятиугольник
`h`	Шестиугольник
Тип линии	
`-`	Сплошная
`:`	Двойной пунктир
`-.`	Штрих-пунктир
`--`	Штриховая

Таким образом, с помощью строковой константы `S` можно изменять цвет линии, представлять узловые точки различными отметками (точка, окружность, крест, треугольник с разной ориентацией вершины и т. д.) и менять тип линии графика.

- `plot(X1,Y1,S1,X2,Y2,S2,X3,Y3,S3,…)` – эта команда строит на одном графике ряд линий, представленных данными вида (X_i, Y_i, S_i), где X_i и Y_i – векторы или матрицы, а S_i – строки. С помощью такой конструкции возможно построение, например, графика функции линией, цвет которой отличается от цвета узловых точек. Так, если надо построить график функции линией синего цвета с красными точками, то вначале надо задать построение графика с точками красного цвета (без линии), а затем графика только линии синего цвета (без точек).

При отсутствии указания на цвет линий и точек он выбирается автоматически из таблицы цветов (белый исключается). Если линий больше шести, то выбор цветов повторяется. Для монохромных систем линии выделяются стилем.

Рассмотрим пример простой программы для построения графиков трех функций с различным стилем представления каждой из них:

```
% Программа построения графиков трех функций
x=-2*pi:0.1*pi:2*pi;
y1=sin(x); y2=sin(x).^2; y3=sin(x).^3;
plot(x,y1,'-m',x,y2,'-.+r',x,y3,'-ok')
```

Эта программа является типичным скрипт-файлом. Графики функций при ее запуске (указанием заданного имени) показаны на рис. 6.3.

Здесь график функции `y1` строится сплошной фиолетовой линией, график `y2` строится штрих-пунктирной линией с точками в виде знака «плюс» красного цве-

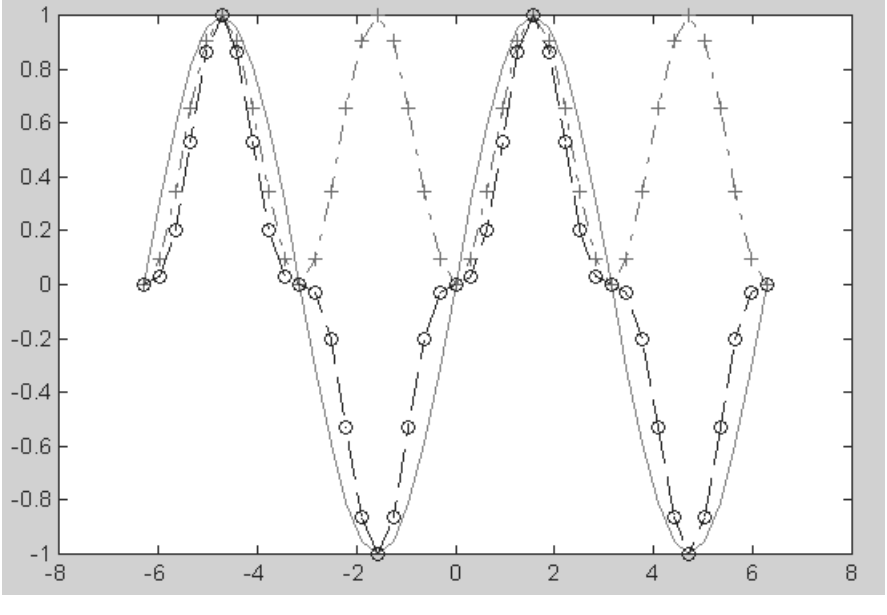

Рис. 6.3. Построение графиков трех функций на одном рисунке с разным стилем линий

та, а график y3 строится штриховой линией с кружками черного цвета. К сожалению, на черно-белых рисунках этой книги вместо разных цветов видны разные градации серого цвета.

6.1.2. Графики в логарифмическом масштабе

Для построения графиков функций со значениями x и y, изменяющимися в широких пределах, нередко используются логарифмические масштабы. Рассмотрим команды, которые используются в таких случаях.

- `loglog(…)` – синтаксис команды аналогичен ранее рассмотренному для функции `plot(…)`. Логарифмический масштаб используется для координатных осей X и Y. Ниже дан пример применения данной команды:

```
>> x=logspace(-1,3); loglog(x,exp(x)./x); grid on
```

На рис. 6.4 представлен график функции $\exp(x)/x$ в логарифмическом масштабе. Обратите внимание на то, что командой `grid on` строится координатная сетка.

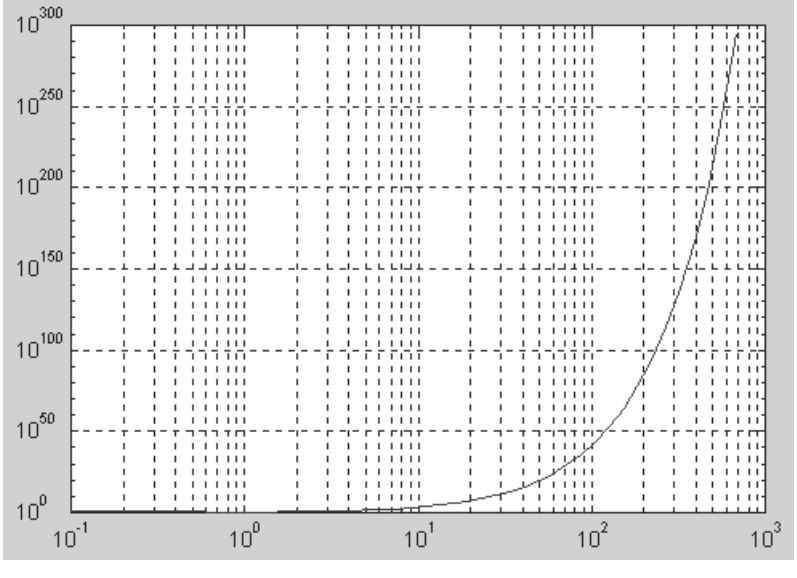

Рис. 6.4. График функции exp(x)/x в логарифмическом масштабе

Неравномерное расположение линий координатной сетки указывает на логарифмический масштаб осей.

6.1.3. Графики в полулогарифмическом масштабе

В некоторых случаях предпочтителен *полулогарифмический* масштаб графиков, когда по одной оси задается логарифмический масштаб, а по другой – линейный. Для построения графиков функций в полулогарифмическом масштабе используются следующие команды:

- `semilogx(…)` строит график функции в логарифмическом масштабе (основание 10) по оси *X* и линейном по оси *Y*;
- `semilogy(…)` строит график функции в логарифмическом масштабе по оси *Y* и линейном по оси *X*.

Запись параметров `(…)` выполняется по аналогии с функцией `plot(…)`. Ниже приводится пример построения графика экспоненциальной функции:

```
>> x=0:0.5:10; semilogy(x,exp(x))
```

График функции при логарифмическом масштабе по оси *Y* представлен на рис. 6.5.

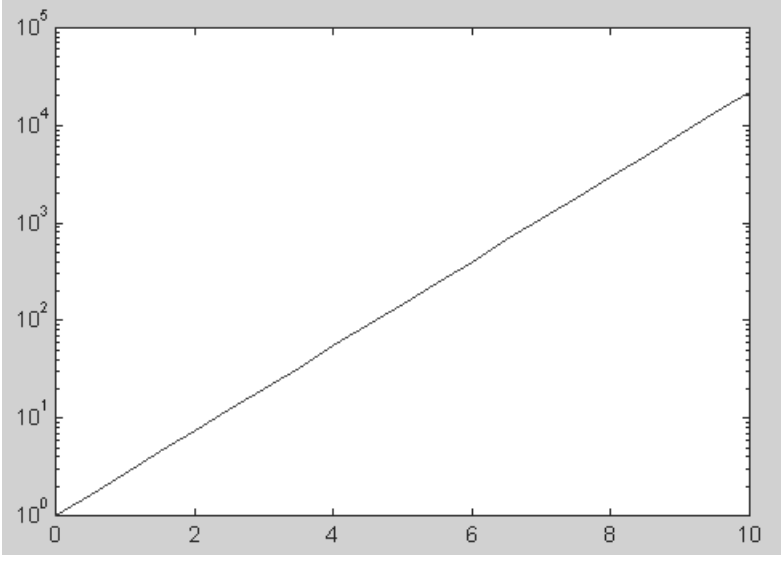

Рис. 6.5. График экспоненты в полулогарифмическом масштабе

Нетрудно заметить, что при таком масштабе график экспоненциальной функции выродился в прямую линию. Масштабной сетки теперь уже нет.

6.1.4. Столбцовые диаграммы

Столбцовые диаграммы широко используются в литературе, посвященной финансам и экономике, а также в математической литературе. Ниже представлены команды для построения таких диаграмм.

- `bar(x,Y)` строит столбцовый график элементов вектора `Y` (или группы столбцов для матрицы Y) со спецификацией положения столбцов, заданной значениями элементов вектора `x`, которые должны идти в монотонно возрастающем порядке;
- `bar(Y)` строит график значений элементов матрицы `Y` так же, как указано выше, но фактически для построения графика используется вектор `x=1:m`;
- `bar(x,Y,WIDTH)` или `BAR(Y,WIDTH)` – команда аналогична ранее рассмотренным, но со спецификацией ширины столбцов (при `WIDTH > 1` столбцы перекрываются). По умолчанию задано `WIDTH = 0.8`.

Возможно применение этих команд и в следующем виде:

```
bar(…,'Спецификация')
```

для задания спецификации графиков, например типа линий, цвета и т. д., по аналогии с командой `plot`. Спецификация 'stacked' задает рисование всех *n* столбцов друг на друге.

Пример построения столбцовой диаграммы матрицы размером 12×3 приводится ниже:

```
>> subplot(2,1,1), bar(rand(12,3),'stacked'), colormap(cool)
```

На рис. 6.6 представлен полученный график.

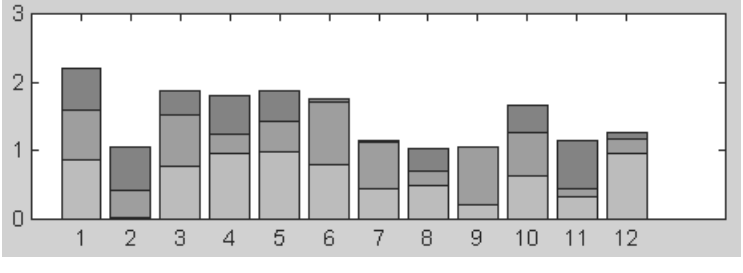

Рис. 6.6. Пример построения диаграммы с вертикальными столбцами

Помимо команды `bar(…)`, существует аналогичная ей по синтаксису команда `barh(…)`, которая строит столбцовые диаграммы с горизонтальным расположением столбцов. Пример, приведенный ниже, дает построения, показанные на рис. 6.7.

```
>> subplot(2,1,1), barh(rand(5,3),'stacked'), colormap(cool)
```

Какое именно расположение столбцов выбрать, зависит от пользователя, использующего эти команды для представления своих данных.

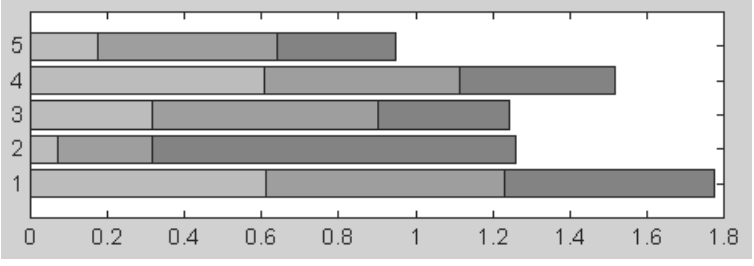

Рис. 6.7. Пример построения столбцовой диаграммы с горизонтальными столбцами

6.1.5. Гистограммы

Классическая *гистограмма* характеризует графически числа попаданий значений элементов вектора `Y` в `M` интервалов с представлением этих чисел в виде столбцовой диаграммы. Для получения данных для гистограммы служит функция `hist`, записываемая в следующем виде:

- `N=hist(Y)` возвращает вектор чисел попаданий для 10 интервалов, выбираемых автоматически. Если `Y` – матрица, то выдается массив данных о числе попаданий для каждого из ее столбцов;
- `N=hist(Y,M)` аналогична вышерассмотренной, но используется `M` интервалов (`M` – скаляр);
- `N=hist(Y,X)` возвращает числа попаданий элементов вектора `Y` в интервалы, центры которых заданы элементами вектора `X`;
- `[N,X]=HIST(…)` возвращает числа попаданий в интервалы и данные о центрах интервалов.

Команда `hist(…)` с синтаксисом, аналогичным приведенному выше, строит график гистограммы. В следующем примере строится гистограмма для 1000 случайных чисел и выводится вектор с данными о числах их попаданий в интервалы, заданные вектором `x`:

```
>> x=-3:0.2:3; y=randn(1000,1);
>> hist(y,x); h=hist(y,x)
h =
   Columns 1 through 12
   0     0     3     7     8     9     11    23    33    43    57    55
   Columns 13 through 24
   70    62    83    87    93    68    70    65    41    35    27    21
   Columns 25 through 31
   12    5     6     3     2     1     0
```

Построенная гистограмма показана на рис. 6.8.

Нетрудно заметить, что распределение случайных чисел близко к нормальному закону. Увеличив их количество, можно наблюдать еще большее соответствие этому закону.

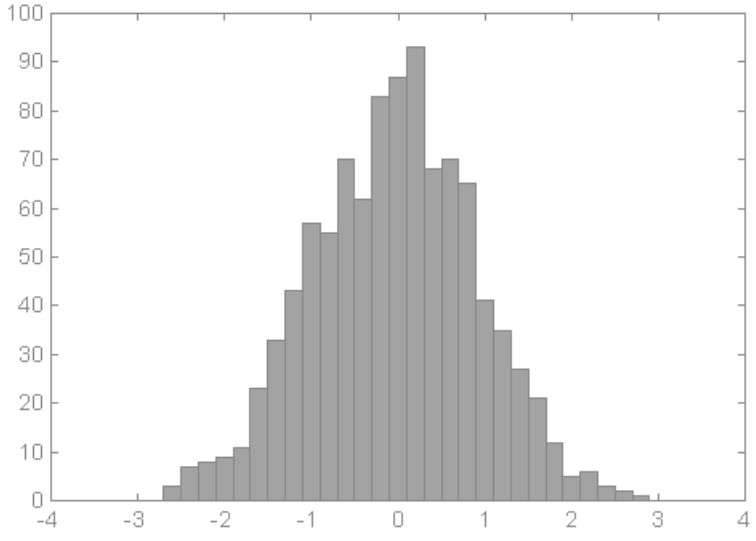

Рис. 6.8. Пример построения гистограммы

6.1.6. Лестничные графики

Лестничные графики визуально представляют собой ступеньки с огибающей, представленной функцией $y(x)$. Такие графики используются, например, для отображения процессов квантования функции $y(x)$, представленной рядом своих отсчетов. При этом в промежутках между отсчетами значения функции считаются постоянными и равными величине последнего отсчета.

Для построения лестничных графиков в системе MATLAB используются команды группы `stairs`:

- `stairs(Y)` строит лестничный график по данным вектора `Y`;
- `stairs(X,Y)` строит лестничный график по данным вектора `Y` с координатами x переходов от ступеньки к ступеньке, заданными значениями элементов вектора `X`;
- `stairs(…,S)` аналогична по действию вышеописанным командам, но строит график линиями, стиль которых задается строками `S`.

Следующий пример иллюстрирует построение лестничного графика:

```
>>  x=0:0.25:10; stairs(x,x.^2);
```

Результат построения представлен на рис. 6.9.

Обратите внимание на то, что отсчеты берутся через равные промежутки по горизонтальной оси. Если, к примеру, отображается функция времени, то `stairs` имеет вид квантованной по времени функции.

Функция `H=stairs(X,Y)` возвращает вектор дескрипторов графических объектов. Функция

```
[XX,YY]=stairs(X,Y)
```

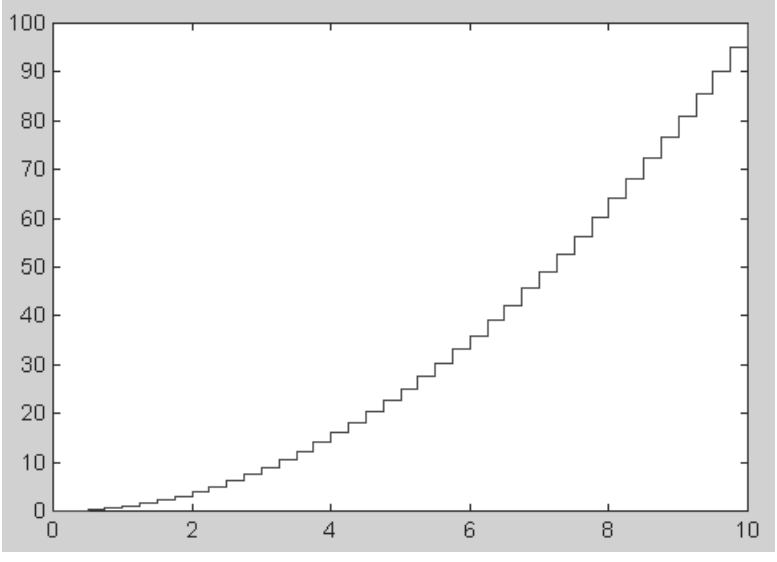

Рис. 6.9. Лестничный график функции x^2

сама по себе графика не строит, а возвращает векторы `XX` и `YY`, которые позволяют построить график с помощью команды `plot(XX,YY)`.

6.1.7. Графики с зонами погрешности

Если данные для построения функции определены с заметной погрешностью, то используют графики функций типа `errorbar` с оценкой погрешности каждой точки путем ее представления в виде буквы *I*, высота которой соответствует заданной погрешности представления точки. Команда `errorbar` используется в следующем виде:

- `errorbar(X,Y,L,U)` строит график значений элементов вектора `Y` в зависимости от данных, содержащихся в векторе `X`, с указанием нижней и верхней границ значений, заданных в векторах `L` и `U`;
- `errorbar(X,Y,E)` и `errorbar(Y,E)` строит графики функции `Y(X)` с указанием этих границ в виде `[Y-E Y+E]`, где `E` – погрешность;
- `errorbar(…,'LineSpec')` аналогична описанным выше командам, но позволяет строить линии со спецификацией `'LineSpec'`, аналогичной спецификации, примененной в команде `plot`.

Следующий пример иллюстрирует применение команды `errorbar`:

```
>> x=-2:0.1:2; y=erf(x);
>> e = rand(size(x))/10; errorbar(x,y,e);
```

Построенный график показан на рис. 6.10.

Функция, записываемая в виде `H=ERRORBAR(…)`, возвращает вектор дескрипторов графических объектов.

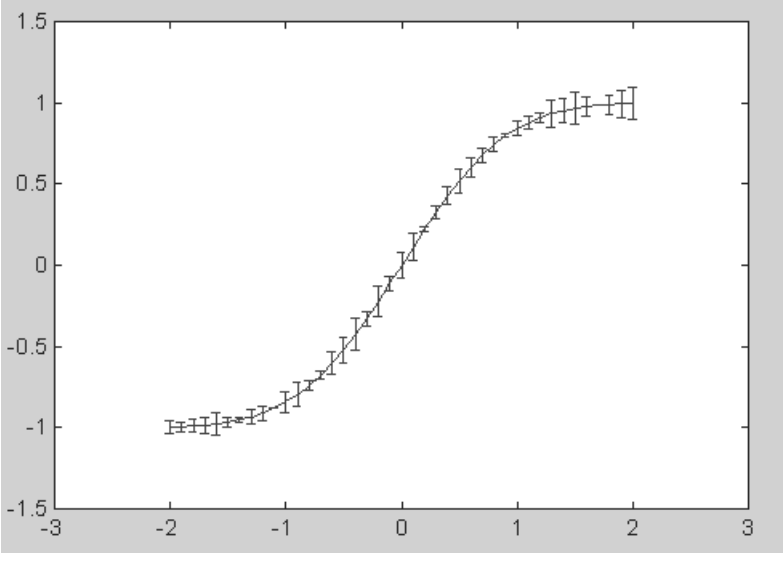

Рис. 6.10. График функции erf(x) с зонами погрешности

6.1.8. Графики дискретных отсчетов функции

Еще один вид графика функции $y(x)$ – ее представление дискретными отсчетами. Этот вид графика применяется, например, при описании квантования сигналов. Каждый отсчет представляется вертикальной чертой, увенчанной кружком, причем высота черты соответствует y-координате точки.

Для построения графика подобного вида используются команды `stem(…)`:

- `stem(Y)` строит график функции с ординатами в векторе `Y` в виде отсчетов;
- `stem(X,Y)` строит график отсчетов с ординатами в векторе `Y` и абсциссами в векторе `X`;
- `stem(…,'filled')` строит график функции с закрашенными маркерами;
- `stem(…,'LINESPEC')` дает построения, аналогичные ранее приведенным командам, но со спецификацией линий `'LINESPEC'`, подобной спецификации, приведенной для функции `plot`.

Следующий пример иллюстрирует применение команды `stem`:

```
>> x = 0:0.1:4; y = sin(x.^2).*exp(-x); stem(x,y)
```

Полученный для данного примера график показан на рис. 6.11.

Функция `H=STEM(…)` строит график и возвращает вектор дескрипторов графических объектов.

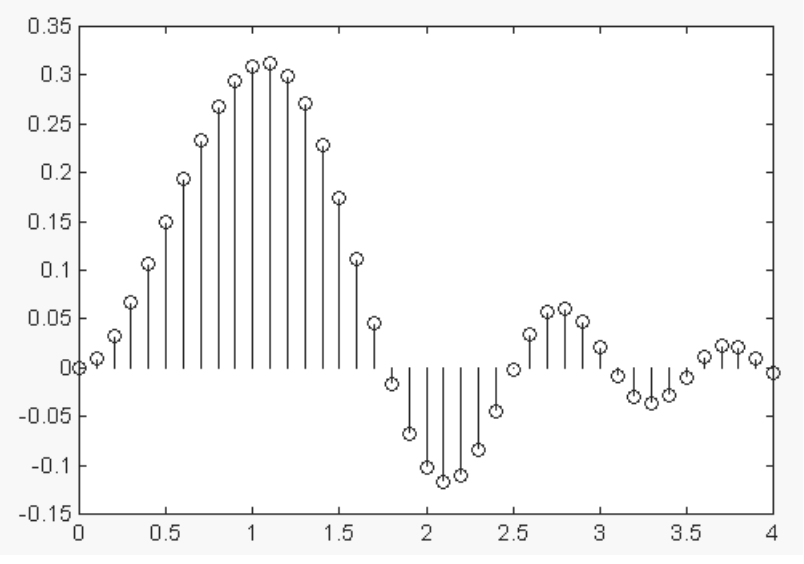

Рис. 6.11. График дискретных отсчетов функции

6.2. Визуализация в полярной системе координат

6.2.1. Графики в полярной системе координат

В *полярной системе координат* любая точка представляется как конец радиус-вектора, исходящего из начала системы координат, имеющего длину `RHO` и угол `THETA`. Для построения графика функции `RHO(THETA)` используются приведенные ниже команды. Угол `THETA` обычно меняется от `0` до `2*pi`. Для построения графиков функций в полярной системе координат используются команды типа `polar(…)`:

- `polar(THETA, RHO)` строит график в полярной системе координат, представляющий собой положение конца радиус-вектора с длиной `RHO` и углом `THETA`;
- `polar(THETA,RHO,S)` аналогична предыдущей команде, но позволяет задавать стиль построения с помощью строковой константы `S` по аналогии с командой `plot`.

Рисунок 6.12 демонстрирует результат выполнения команд:

```
>> t=0:pi/50:2*pi; polar(t,sin(5*t))
```

Графики функций в полярных координатах могут иметь весьма разнообразный вид, порой напоминая такие объекты природы, как снежинки или кристалли-

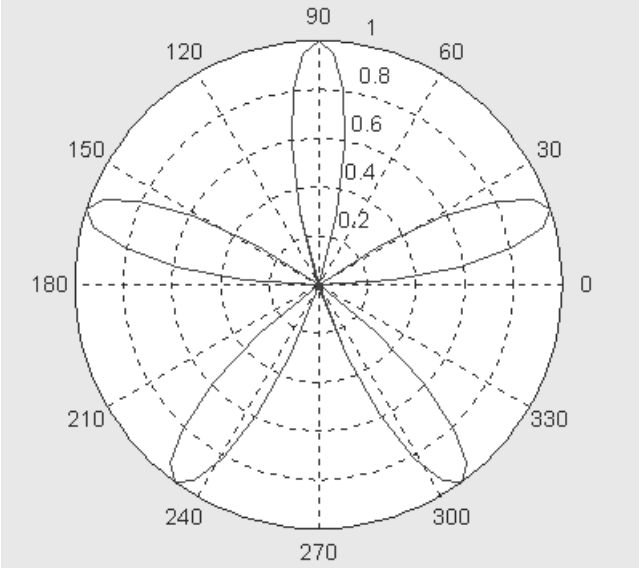

Рис. 6.12. График функции в полярной системе координат

ки льда на стекле. Вы можете сами попробовать построить несколько таких графиков – многие получают от этого удовольствие.

6.2.2. Угловые гистограммы

Угловые гистограммы находят применение в индикаторах радиолокационных станций, для отображения «роз» ветров и при построении других специальных графиков. Для этого используется ряд команд типа `rose(…)`:

- `rose(THETA)` строит угловую гистограмму для 20 интервалов по данным вектора `THETA`;
- `rose(THETA,N)` строит угловую гистограмму для `N` интервалов в пределах угла от `0` до `2*pi` по данным вектора `THETA`;
- `rose(THETA,X)` строит угловую гистограмму по данным вектора `THETA` со спецификацией интервалов, указанной в векторе `X`.

Следующий пример иллюстрирует применение команды `rose`:

```
>> rose(1:100,12)
```

На рис. 6.13 показан пример построения графика командой `rose`.

Функция `H=rose(…)` строит график и возвращает вектор дескрипторов графических объектов, а функция `[T,R]=rose(…)` сама по себе график не строит, но возвращает векторы `T` и `R`, которые нужны команде `polar(T,R)` для построения подобной гистограммы.

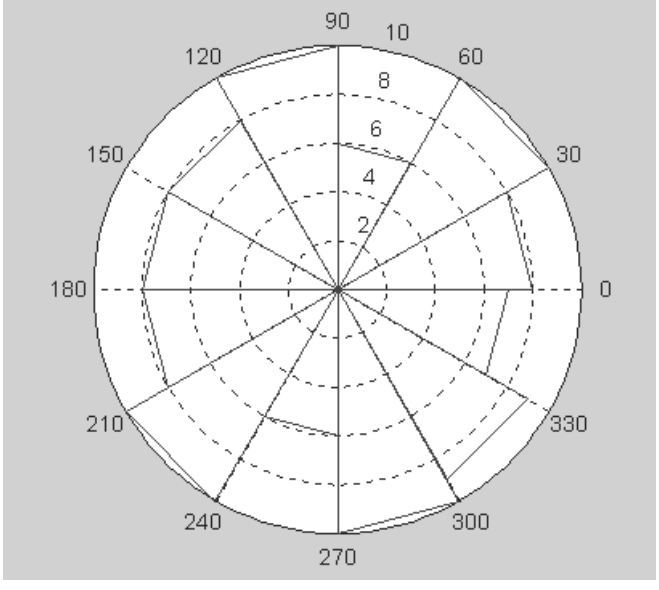

Рис. 6.13. Угловая гистограмма

6.3. Визуализация векторов

6.3.1. Графики векторов

Иногда желательно представление ряда радиус-векторов в их обычном виде, то есть в виде стрелок, исходящих из начала координат и имеющих угол и длину, определяемые действительной и мнимой частями комплексных чисел, представляющих эти векторы. Для этого служит группа команд `compass`:

- `compass(U,V)` строит графики радиус-векторов с компонентами `(U,V)`, представляющими действительную и мнимую части каждого из радиус-векторов;
- `compass(Z)` эквивалентно `compass(real(Z), imag(Z))`;
- `compass(U,V,LINESPEC)` и `compass(Z,LINESPEC)` аналогичны представленным выше командам, но позволяют задавать спецификацию линий построения `LINESPEC`, подобную описанной для команды `plot`.

В следующем примере показано использование команды `compass`:

```
>> Z=[-1+2i,-2-3i,2+3i,5+2i]; compass(Z)
```

Построенный в этом примере график представлен на рис. 6.14.

Функция `H=COMPASS(...)` строит график и возвращает дескрипторы графических объектов.

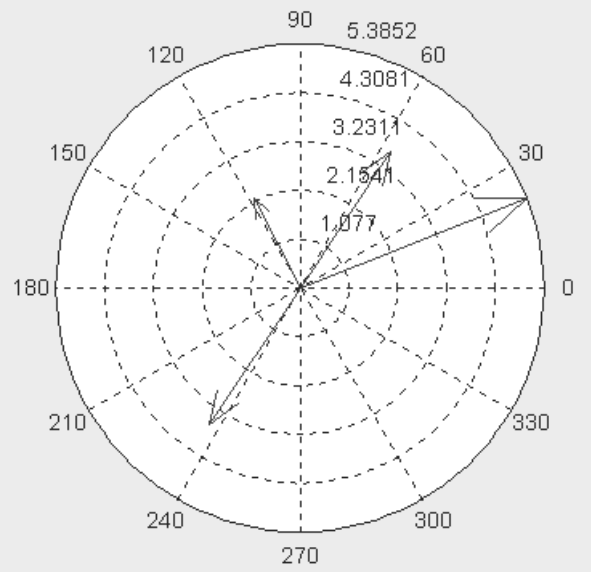

Рис. 6.14. Построение радиус-векторов

6.3.2. График проекций векторов на плоскость

Иногда полезно отображать комплексные величины вида $z = x + yi$ в виде проекции радиус-вектора на плоскость. Для этого используется семейство графических команд класса `feather`:

- `feather(U,V)` строит график проекции векторов, заданных компонентами `U` и `V`, на плоскость;
- `feather(Z)` для вектора `Z` с комплексными элементами дает построения, аналогичные `feather(REAL(Z),IMAG(Z))`;
- `feather(…, S)` дает построения, описанные выше, но со спецификацией линий, заданной строковой константой `S` по аналогии с командой `plot`.

Пример применения команды `feather`:

```
>> x=0:0.1*pi:3*pi; y=0.05+i; z=exp(x*y); feather(z)
```

График, построенный в этом последнем примере, показан на рис. 6.15.

Функция `H=FEATHER(…)` строит график и возвращает вектор дескрипторов графических объектов.

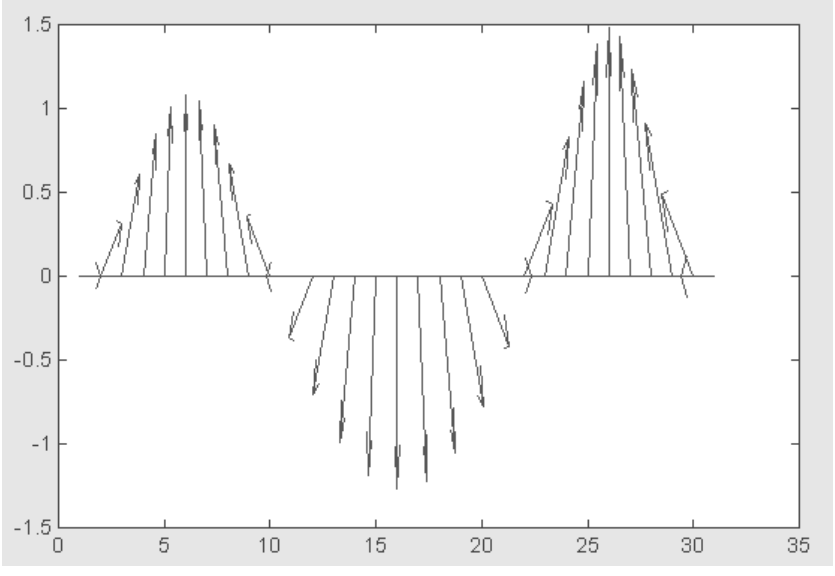

Рис. 6.15. График, построенный командой feather

6.4. Основы трехмерной графики

6.4.1. Контурные графики

Контурные графики служат для представления на плоскости функции двух переменных вида $z(x,y)$ с помощью линий равного уровня. Они получаются, если трехмерная поверхность пересекается рядом плоскостей, расположенных параллельно друг другу. При этом контурный график представляет собой совокупность спроецированных на плоскость (x,y) линий пересечения поверхности $z(x,y)$ плоскостями. Типичный пример контурных графиков – обычные карты.

Для построения контурных графиков используются команды `contour`:

- `contour(Z)` строит контурный график по данным матрицы `Z` с автоматическим заданием диапазонов изменения x и y;
- `contour(X,Y,Z)` строит контурный график по данным матрицы `Z` с указанием спецификаций для `X` и `Y`;
- `contour(Z,N)` и `contour(X,Y,Z,N)` дает построения, аналогичные ранее описанным командам, с заданием `N` линий равного уровня (по умолчанию `N=10`);
- `contour(Z,V)` и `contour(X,Y,Z,V)` строят линии равного уровня для высот, указанных значениями элементов вектора `V`;
- `contour(Z,[v v])` или `contour(X,Y,Z,[v v])` вычисляет одиночный контур для уровня `v`;

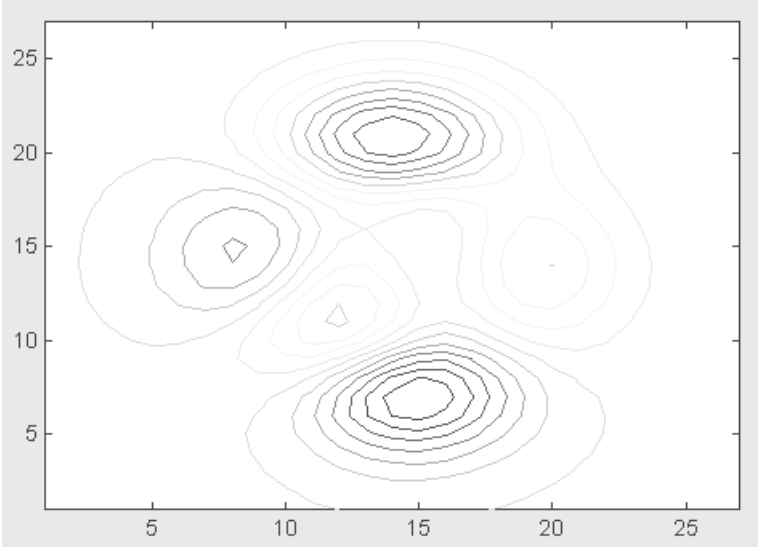

Рис. 6.16. Контурный график, построенный с помощью команды contour

- `[C,H] = contour(...)` возвращает дескрипторы – матрицу **C** и вектор-столбец **H**. Они могут использоваться как входные параметры для команды `clabel`;
- `contour(...,'LINESPEC')` позволяет использовать перечисленные выше команды с указанием спецификации линий, которыми идет построение.

Пример построения контурного графика поверхности, заданной функцией `peaks`:

```
>>  z=peaks(27); contour(z,15)
```

Построенный в этом примере график показан на рис. 6.16. Заметим, что объект – функция `peaks` – задан в системе в готовом виде.

Графики этого типа часто используются в топографии для представления на листе бумаги (как говорят математики – на плоскости) объемного рельефа местности. Для оценки высот контурных линий используется их функциональная окраска.

6.4.2. Создание массивов данных для трехмерной графики

Поверхности как объекты трехмерной графики обычно описываются функцией двух переменных $z(x,y)$. Специфика построения трехмерных графиков требует не просто задания ряда значений x и y, то есть векторов `x` и `y`. Она требует определения для `X` и `Y` двумерных массивов – матриц. Для создания таких массивов слу-

жит функция meshgrid. В основном она используется совместно с функциями построения графиков трехмерных поверхностей. Функция meshgrid записывается в следующих формах:

- [X,Y] = meshgrid(x,y) преобразует область, заданную векторами x и y, в массивы X и Y, которые могут быть использованы для вычисления функции двух переменных и построения трехмерных графиков. Строки выходного массива X являются копиями вектора x; а столбцы Y – копиями вектора y;
- [X,Y] = meshgrid(x) аналогична [X,Y] = meshgrid(x,x);
- [X,Y,Z] = meshgrid(x,y,z) возвращает трехмерные массивы, используемые для вычисления функций трех переменных и построения трехмерных графиков.

  Пример:

```
>> [X,Y] = meshgrid(1:4,13:17)
X =
        1     2     3     4
        1     2     3     4
        1     2     3     4
        1     2     3     4
        1     2     3     4
Y =
       13    13    13    13
       14    14    14    14
       15    15    15    15
       16    16    16    16
       17    17    17    17
```

  Приведем еще один пример применения функции meshgrid:

```
>> [X,Y] = meshgrid(-2:.2:2, -2:.2:2);
```

  Такой вызов функции позволяет задать опорную плоскость для построения трехмерной поверхности при изменении x и y от –2 до 2 с шагом 0,2. Дополнительные примеры применения функции meshgrid будут приведены далее при описании соответствующих команд. Рекомендуется ознакомиться также с командами surf и slice (ломтик).

Функция ndgrid является многомерным аналогом функции meshgrid:

- [X1,X2,X3,…] = ndgrid(x1,x2,x3,…) преобразует область, заданную векторами x1,x2,x3…, в массивы X1,X2,X3…, которые могут быть использованы для вычисления функций нескольких переменных и многомерной интерполяции. i-я размерность выходного массива Xi является копией вектора xi;
- [X1,X2,…] = ndgrid(x) аналогична [X1,X2,…] = ndgrid(x,x,…).

  Пример применения функции ndgrid представлен ниже:

```
[X1,X2] = ndgrid(-2:.2:2, -2:.2:2);
Z = X1 .* exp(-X1.^2 – X2.^2); mesh(Z)
```

Рекомендуем читателю опробовать действие этого примера.

6.4.3. Графики поля градиентов

Для построения *графиков полей градиента* служат команды `quiver`:

- `quiver(X,Y,U,V)` строит график поля градиентов в виде стрелок для каждой пары элементов массивов `X` и `Y`, причем элементы массивов `U` и `V` указывают направление и размер стрелок;
- `quiver(U,V)` строит векторы скорости в равнорасположенных точках на плоскости (*x*,*y*);
- `quiver(U,V,S)` или `quiver(X,Y,U,V,S)` автоматически масштабирует стрелки по сетке и затем вытягивает их по значению `S`. Используйте `S=0`, чтобы построить стрелки без автоматического масштабирования;
- `quiver(…,LINESPEC)` использует для векторов указанный тип линии. Указанные в `LINESPEC` маркеры рисуются у оснований, а не на концах векторов. Для отмены любого вида маркера используйте спецификацию `'.'`. Спецификации линий, цветов и маркеров были подробно описаны в разделе, посвященном команде `plot`;
- `quiver(…,'filled')` дает график с закрашенными маркерами;
- `H=quiver(…)` строит график и возвращает вектор дескрипторов.

Ниже представлен пример программы с применением команды `quiver`:

```
% Программа построения графика поля градиентов
x = -2:.2:2; y = -1:.2:1;
[xx,yy] = meshgrid(x,y);
zz = xx.*exp(-xx.^2-yy.^2);
[px,py] = gradient(zz,.2,.2);
quiver(x,y,px,py,2);
```

Построенный с помощью этой программы график показан на рис. 6.17.

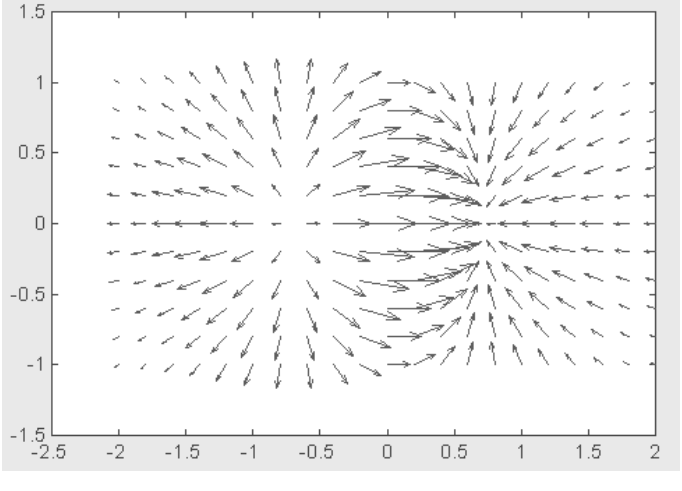

Рис. 6.17. Пример построения графика поля градиентов

Нетрудно заметить, что представление поля градиентов стрелками дает весьма наглядное представление о линиях поля, указывая области, куда эти линии впадают и откуда они исходят.

6.4.4. Графики поверхностей

Команда `plot3(...)` является аналогом команды `plot(...)`, но относится к функции двух переменных $z(x,y)$. Она строит аксонометрическое изображение трехмерных поверхностей и представлена следующими формами:

- `plot3(x,y,z)` строит массив точек, представленных векторами `x`, `y` и `z`, соединяя их отрезками прямых. Эта команда имеет ограниченное применение;
- `plot3(X,Y,Z)`, где `X`, `Y` и `Z` – три матрицы одинакового размера, строит точки с координатами `X(i,:)`, `Y(i,:)` и `Z(i,:)` и соединяет их отрезками прямых.

  Ниже дан пример программы построения трехмерной поверхности, описываемой функцией `z(x,y)=x^2+y^2`:

```
% Программа построения поверхности линиями
[X,Y]=meshgrid([-3:0.15:3]);
Z=X.^2+Y.^2;
plot3(X,Y,Z)
```

  График этой поверхности показан на рис. 6.18.

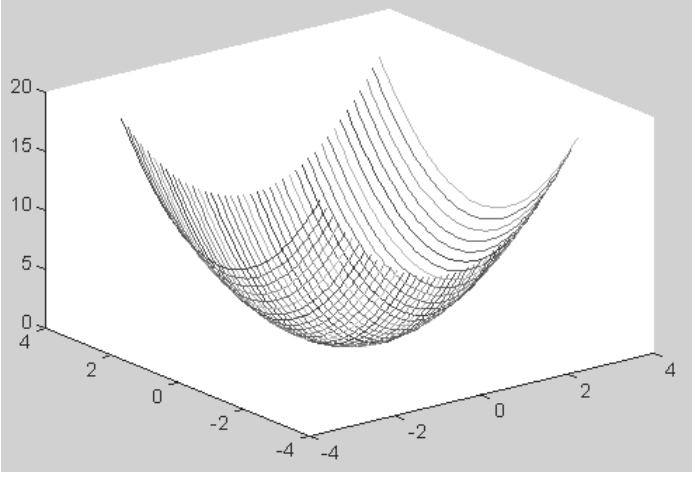

Рис. 6.18. График поверхности, построенный линиями

- `plot3(X,Y,Z,S)` обеспечивает построения, аналогичные рассмотренным ранее, но со спецификацией стиля линий и точек, соответствующей спецификации команды `plot`. Ниже дан пример применения этой команды для построения поверхности кружками:

```
% Программа построения поверхности кружками
[X,Y]=meshgrid([-3:0.15:3]);
Z=X.^2+Y.^2;
plot3(X,Y,Z,'o')
```

График поверхности, построенный кружками, показан на рис. 6.19.

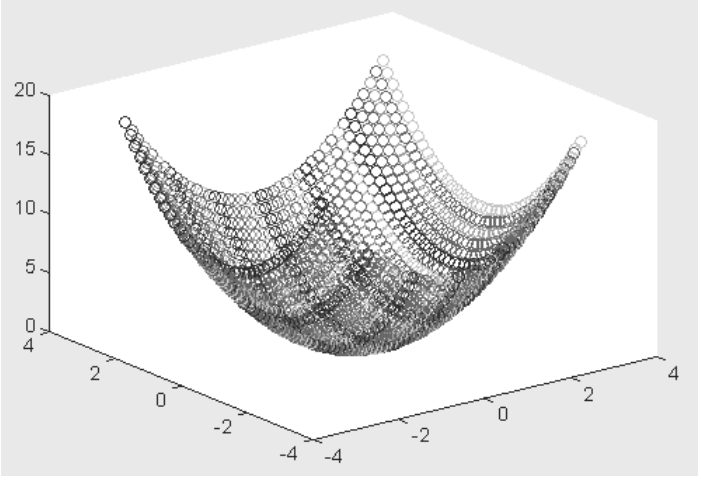

Рис. 6.19. График поверхности, построенный разноцветными кружками

- `plot3(x1,y1,z1,s1,x2,y2,z2,s2,x3,y3,z3,s3,…)` строит на одном рисунке графики нескольких функций `z1(x1,y1)`, `z2(x2,y2)` и т. д. со спецификацией линий и маркеров каждой из них.
  Пример применения последней команды дан ниже:

```
% Программа построения сетчатого графика функции
[X,Y]=meshgrid([-3:0.15:3]);
Z=X.^2+Y.^2;
plot3(X,Y,Z,'-k',Y,X,Z,'-k')
```

  График функции, соответствующей последнему примеру, представлен на рис. 6.20.

В данном случае строятся два графика одной и той же функции с взаимно перпендикулярными образующими линиями. Поэтому график имеет вид сетки без окраски ее ячеек (напоминает проволочный каркас фигуры).

6.4.5. Сетчатые 3D-графики с окраской

Наиболее представительными и наглядными являются *сетчатые графики* поверхностей с заданной или функциональной окраской. В названии их команд присутствует слово `mesh`. Имеются три группы таких команд. Ниже приведены данные

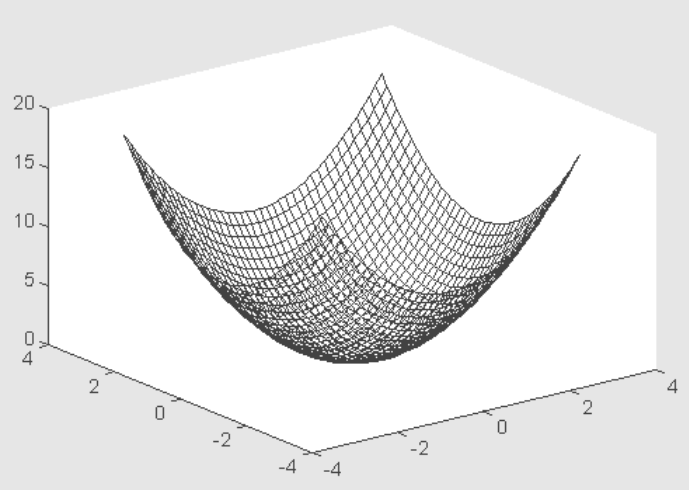

Рис. 6.20. График функции в сетчатом представлении

о наиболее полных формах этих команд. Наличие более простых форм можно уточнить, используя команду `help Имя`, где `Имя` – имя соответствующей команды.

- `mesh(X,Y,Z,C)` выводит в графическое окно сетчатую поверхность `Z(X,Y)` с цветами узлов поверхности, заданными массивом `C`.
- `mesh(X,Y,Z)` – аналог предшествующей команды при `C=Z`. В данном случае используется функциональная окраска, при которой цвет задается высотой поверхности.

Возможны также формы команды `mesh(x,y,Z)`, `mesh(x,y,Z,C)`, `mesh(Z)` и `mesh(Z,C)`. Функция `mesh` возвращает дескриптор для объекта класса `surface`. Ниже приводится пример программы с применением команды `mesh`:

```
% Программа построения графика поверхности с окраской
[X,Y]=meshgrid([-3:0.15:3]);
Z=X.^2+Y.^2;
mesh(X,Y,Z)
```

На рис. 6.21 показан график поверхности, созданной командой `mesh(X,Y,Z)`. Нетрудно заметить, что функциональная окраска линий поверхности заметно усиливает наглядность ее представления.

MATLAB имеет несколько графических функций, возвращающих *матричный образ поверхностей*. Например, функция `peaks(N)` возвращает матричный образ поверхности с рядом пиков и впадин. Такие функции удобно использовать для проверки работы графических команд трехмерной графики. Для упомянутой функции `peaks` можно привести такой пример:

```
>> z=peaks(25); mesh(z);
```

График поверхности, описываемой функцией `peaks`, представлен на рис. 6.22.

Рекомендуется ознакомиться с командами и функциями, используемыми совместно с описанными командами: `axis`, `caxis`, `colormap`, `hold`, `shading` и `view`.

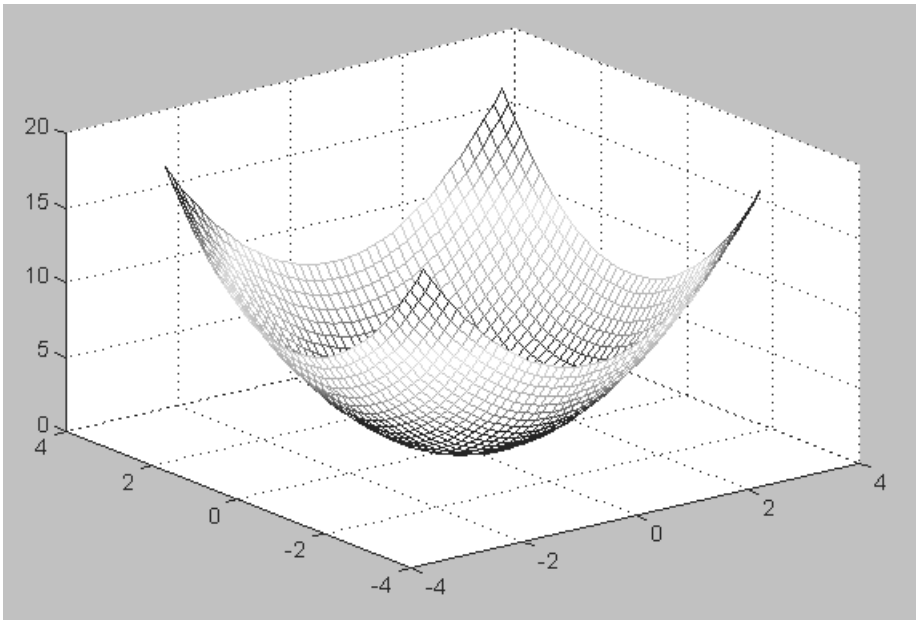

Рис. 6.21. График поверхности, созданный командой mesh(X,Y,Z)

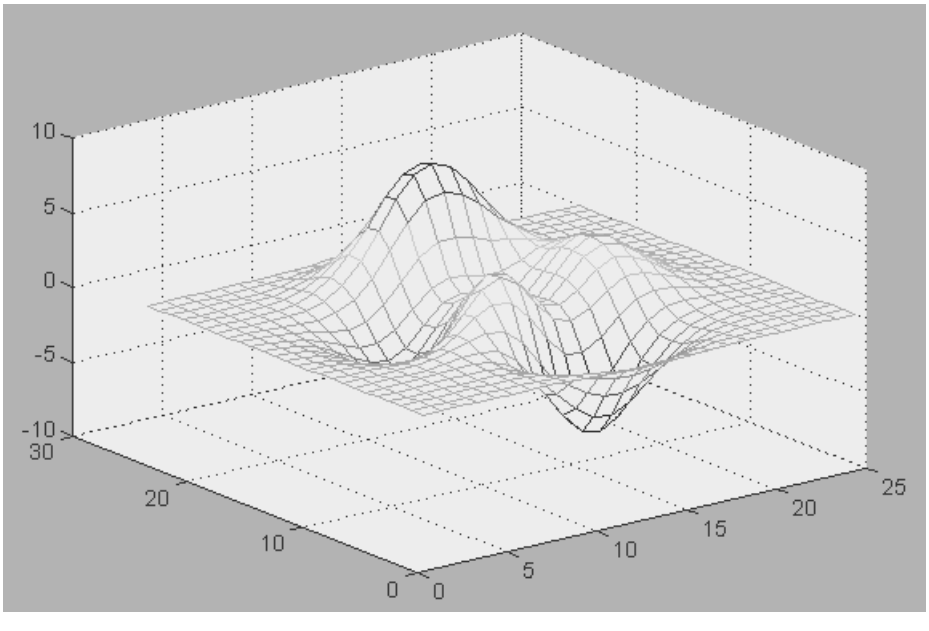

Рис. 6.22. График поверхности, описываемой функцией peaks

6.4.6. Сетчатые 3D-графики с проекциями

Иногда график поверхности полезно объединить с контурным графиком ее проекции на плоскость, расположенным под поверхностью. Для этого используется команда `meshc`:

- `meshc(...)` аналогична `mesh(...)`, но помимо графика поверхности дает изображение ее проекции в виде линий равного уровня (графика типа `contour`).

Ниже дан пример применения этой команды:

```
% Программа построения графика поверхности
% и ее проекции на плоскость
[X,Y]=meshgrid([-3:0.15:3]);
Z=X.^2+Y.^2;
meshc(X,Y,Z)
```

Построенный график показан на рис. 6.23.

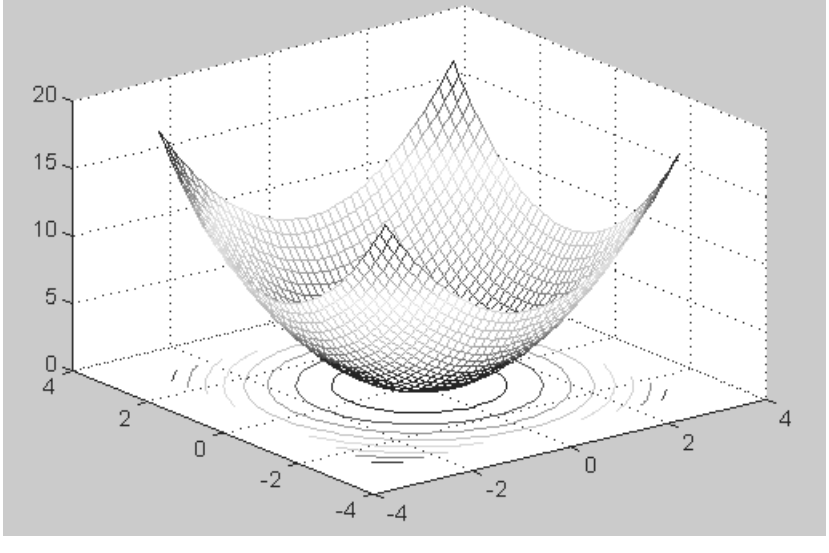

Рис. 6.23. График поверхности и ее проекции на расположенную ниже плоскость

Нетрудно заметить, что график такого типа дает наилучшее представление об особенностях поверхности.

6.4.7. Построение поверхности столбцами

Еще один тип представления поверхности, когда она строится из многочисленных столбцов, дают команды класса `meshz`:

- `meshz(...)` аналогична `mesh(...)`, но строит поверхность как бы в виде столбиков.

Следующая программа иллюстрирует применение команды `mesh`:

```
% Программа построения поверхности столбцами
[X,Y]=meshgrid([-3:0.15:3]);
Z=X.^2+Y.^2; meshz(X,Y,Z)
```

Столбцовый график поверхности показан на рис. 6.24.

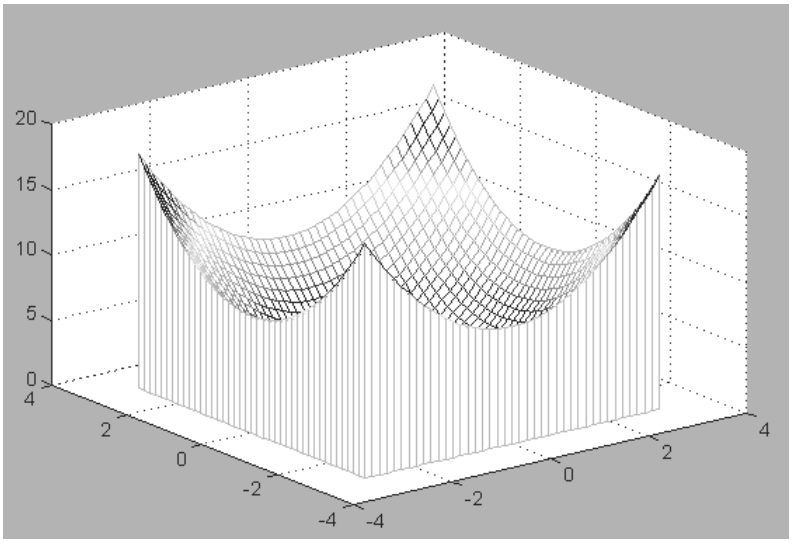

Рис. 6.24. Построение поверхности столбцами

Графики такого типа используются довольно редко. Возможно, он полезен архитекторам или скульпторам, поскольку дает неплохое объемное представление о поверхностях.

6.5. Улучшенные средства визуализации 3D-графики

6.5.1. Построение поверхности с окраской

Особенно наглядное представление о поверхностях дают сетчатые графики, использующие *функциональную закраску* ячеек. Например, цвет окраски поверхности $z(x,y)$ может быть поставлен в соответствии с высотой z поверхности с выбором для малых высот темных тонов, а для больших – светлых.

Для построения таких поверхностей используются команды класса `surf(…)`:

- `surf(X,Y,Z,C)` строит цветную параметрическую поверхность по данным матриц `X`, `Y` и `Z` с цветом, задаваемым массивом `C`;
- `surf(X,Y,Z)` аналогична предшествующей команде, где `C=Z`, так что цвет задается высотой той или иной ячейки поверхности;
- `surf(x,y,Z)` и `surf(x,y,Z,C)` с двумя векторными аргументами `x` и `y` – векторы `x` и `y` заменяют два первых матричных аргумента и должны иметь длины `length(x)=n` и `length(y)=m`, где `[m,n] = size(Z)`. В этом случае вершины областей поверхности представлены тройками координат `(x(j), y(i), Z(i,j))`. Заметим, что `x` соответствует столбцам `Z`, а `y` соответствует строкам;
- `surf(Z)` и `surf(Z,C)` используют `x = 1:n` и `y = 1:m`. В этом случае высота `Z` – однозначно определенная функция, заданная геометрически прямоугольной сеткой;
- `h=surf(…)` строит поверхность и возвращает дескриптор объекта класса `surface`.

Команды `axis`, `caxis`, `colormap`, `hold`, `shading` и `view` задают координатные оси и свойства поверхности, которые могут использоваться для большей эффектности показа поверхности или фигуры.

Ниже приведен простой пример построения поверхности – параболоида:

```
% Программа построения графика параболоида с окраской
[X,Y]=meshgrid([-3:0.15:3]);
Z=X.^2+Y.^2;
surf(X,Y,Z)
```

Соответствующий этому примеру график показан на рис. 6.25.

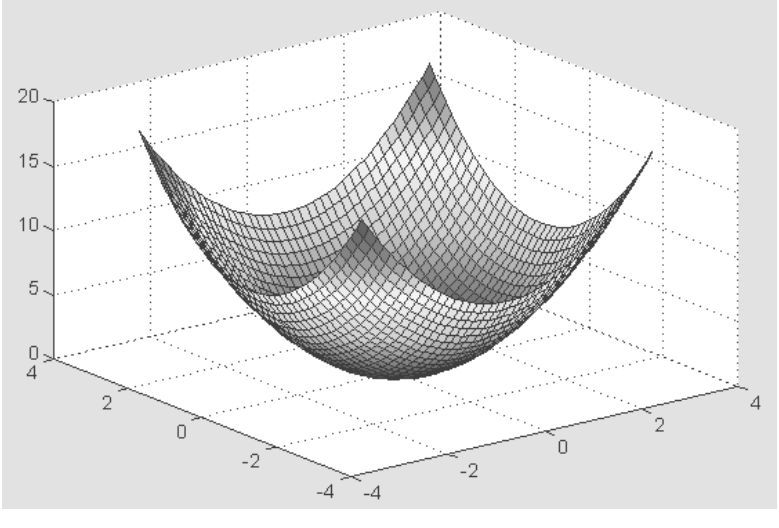

Рис. 6.25. График параболоида с функциональной окраской ячеек

Можно заметить, что благодаря функциональной окраске график поверхности гораздо более выразителен, чем при построениях без такой окраски, представленных ранее (причем даже в том случае, когда цветной график печатается в черно-белом виде).

В следующей программе используется функциональная окраска оттенками серого цвета с выводом шкалы цветовых оттенков:

```
  % Программа построения параболоид
  % со шкалой цветовых оттенков
[X,Y]=meshgrid([-3:0.1:3]);
Z=sin(X)./(X.^2+Y.^2+0.3);
surf(X,Y,Z); colormap(gray)
shading interp; colorbar
```

В этом примере команда `colormap(gray)` задает окраску тонами серого цвета, а команда `shading interp` обеспечивает устранение изображения сетки и задает интерполяцию для оттенков цвета объемной поверхности. На рис. 6.26 показан вид графика, построенного при исполнении этой программы.

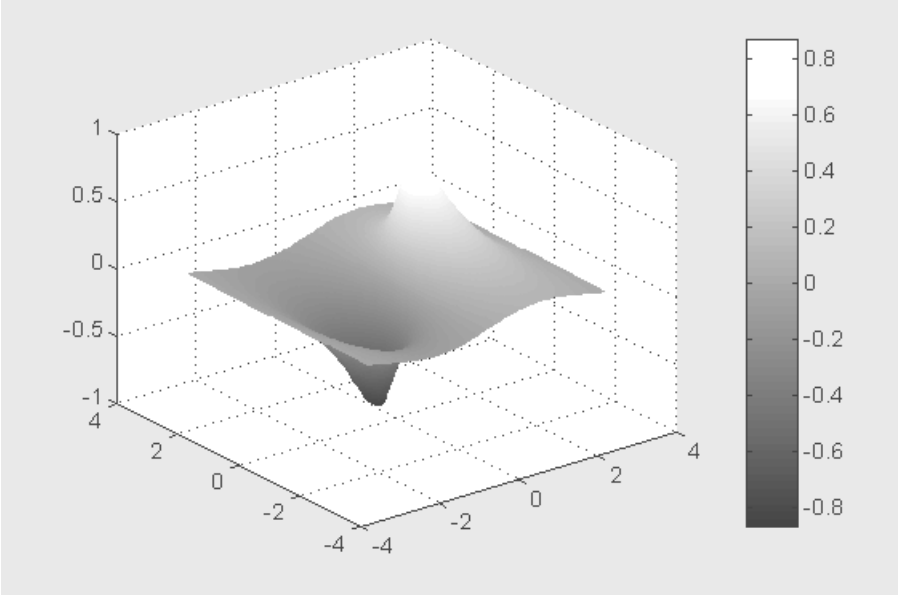

Рис. 6.26. График поверхности с функциональной окраской серым цветом

Обычно применение интерполяции для окраски придает поверхностям и фигурам более реалистичный вид, но фигуры каркасного вида дают более точные количественные данные о каждой точке.

6.5.2. Построение поверхности и ее проекции

Для повышения наглядности представления поверхностей можно использовать дополнительный график линий равного уровня, получаемый путем проецирования поверхности на опорную плоскость графика (под поверхностью). Для этого используется команда `surfc`:

- `surfc(...)` аналогична команде `surf`, но обеспечивает дополнительное построение контурного графика проекции фигуры на опорную плоскость.

Пример применения команды `surfc` приводится ниже:

```
>> [X,Y]=meshgrid([-3:0.1:3]);
>> Z=sin(X)./(X.^2+Y.^2+0.3); surfc(X,Y,Z)
```

На рис. 6.27 показаны графики, построенные в данном примере.

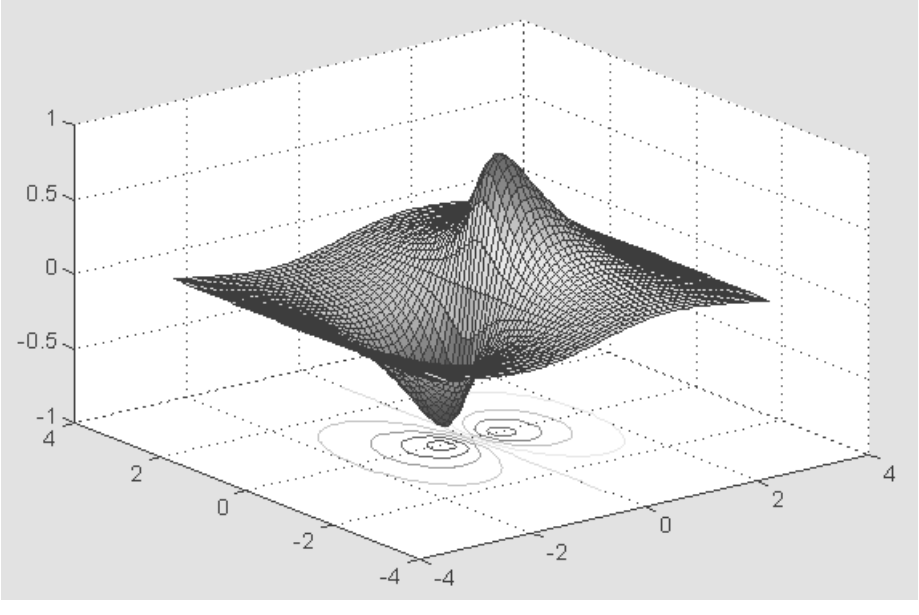

Рис. 6.27. График поверхности и ее проекции на опорную плоскость

Рассмотрим еще один пример применения команды `surfc`, на этот раз для построения поверхности, описываемой функцией `peaks` с применением интерполяции цветов и построением цветовой шкалы:

```
% Программа построения поверхности peaks
% с построением шкалы цветовых оттенков
[X,Y]=meshgrid([-3:0.1:3]);
Z=peaks(X,Y);
```

```
surfc(X,Y,Z)
shading interp;
colorbar
```

Рисунок 6.28 показывает график, построенный при пуске данной программы.

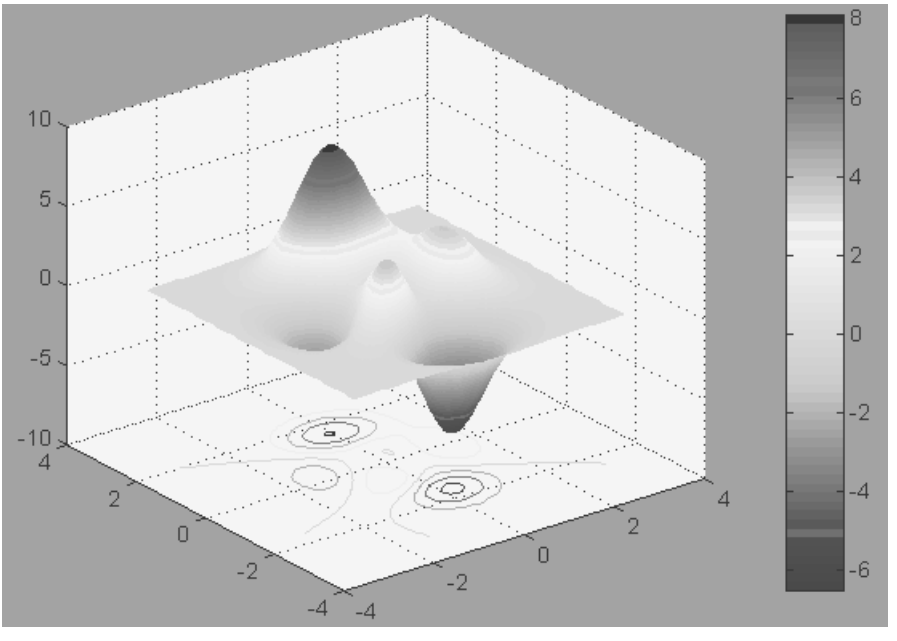

Рис. 6.28. График функции peaks с проекцией и шкалой цветов

И здесь нетрудно заметить, что графики сложных поверхностей с интерполяцией цветовых оттенков выглядят более реалистичными, чем графики сетчатого вида и графики без интерполяции цветов.

6.5.3. Построение освещенной поверхности

Пожалуй, наиболее реалистичный вид имеют графики поверхностей, в которых имитируется освещение от точечного источника света, расположенного в заданном месте координатной системы. Графики имитируют оптические эффекты рассеивания, отражения и зеркального отражения света. Для получения таких графиков используется команда `surfl`:

- `surfl(…)` аналогична команде `surf(…)`, но строит график поверхности с подсветкой от источника света;
- `surfl(Z,S)` или `surfl(X,Y,Z,S)` строит графики поверхности с подсветкой от источника света, положение которого в системе декартовых координат задается вектором `S=[Sx,Sy,Sz]`, а в системе сферических координат – вектором `S=[AZ,EL]`;

- `surfl(…,'light')` позволяет при построении задать цвет подсветки с помощью объекта `Light`;
- `surfl(…,'cdata')` при построении имитирует эффект отражения;
- `surfl(X,Y,Z,S,K)` задает построение поверхности с параметрами, заданными вектором `K=[ka,kd,ks,spread]`, где `ka` – коэффициент фоновой подсветки, `kd` – коэффициент диффузного отражения, `ks` – коэффициент зеркального отражения и `spread` – коэффициент глянцевитости;
- `H=surfl(…)` строит поверхность и возвращает дескрипторы поверхности и источников света.

По умолчанию вектор `S` задает углы азимута и возвышения в 45°. Используя команды `cla`, `hold on`, `view(AZ,EL)`, `surfl(…)` и `hold off`, можно получить дополнительные возможности управления освещением. Надо полагаться на упорядочение точек в `X`, `Y` и `Z` матрицах, чтобы определить внутреннюю и внешнюю стороны параметрических поверхностей. Попробуйте транспонировать матрицы и использовать `surfl(X',Y',Z')`, если вам не понравился результат работы этой команды. Для вычисления векторов нормалей поверхности `surfl` требует в качестве аргументов матрицы с размером по крайней мере 3×3.

Ниже представлен пример применения команды `surfl`:

```
% Программа построения поверхности с имитацией
% ее освещенности от точечного источника
[X,Y]=meshgrid([-3:0.1:3]);
Z=sin(X)./(X.^2+Y.^2+0.3);
surfl(X,Y,Z);
colormap(gray)
shading interp;
colorbar
```

Построенная в этом примере поверхность представлена на рис. 6.29.

Сравните этот рисунок с рис. 6.26, на котором та же поверхность построена без имитации ее освещения.

> ***Примечание***
>
> *Нетрудно заметить определенную логику в названиях графических команд. Имя команды состоит из основного слова и суффикса расширения. Например, все команды построения поверхностей имеют основное слово `surf` (сокращение от surface – поверхность) и суффиксы: `c` – для контурных линий поверхности, `l` – для освещенности и т. д. Это правило облегчает запоминание многочисленных команд графики.*

6.5.4. Средства управления подсветкой и обзором фигур

Рекомендуется с помощью команды `help` ознакомиться с командами, задающими управление подсветкой и связанными с ней оптическими эффектами:

- `diffuse` – задание эффекта диффузионного рассеяния;

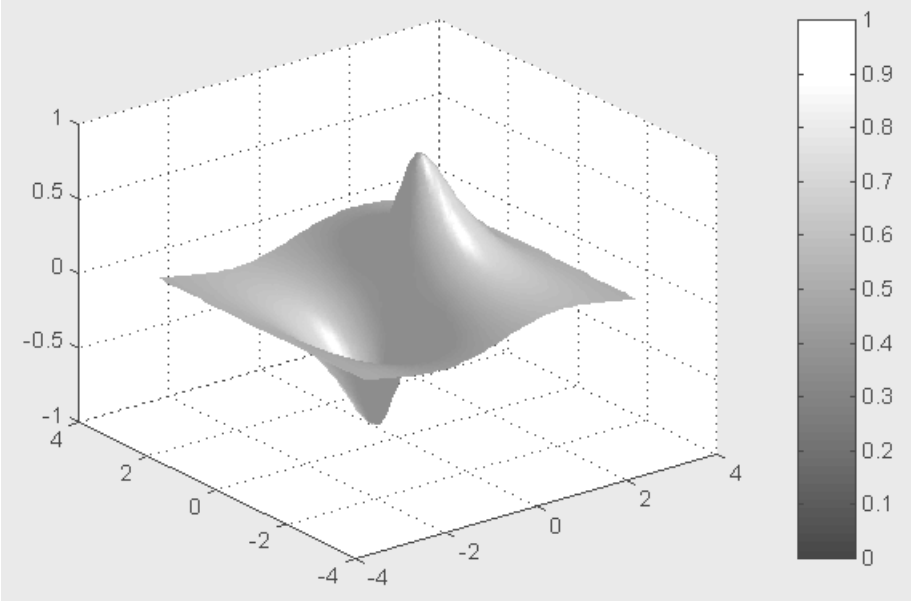

Рис. 6.29. График поверхности с имитацией ее освещения точечным источником

- `lighting` – управление подсветкой;
- `material` – имитация свойств рассеивания света различными материалами;
- `specular` – задание эффекта зеркального отражения.

Следующие три команды позволяют управлять углами просмотра, под которыми рассматривается видимая в графическом окне фигура:

- `view` – задание положения точки просмотра;
- `viewmtx` – задание и вычисление матрицы вращения;
- `rotate3d` – задание поворота трехмерной фигуры.

В ряде случаев применением этих команд можно добиться большей выразительности трехмерных объектов. Скорость построения таких графиков сильно зависит от аппаратной поддержки графики в конкретном ПК. Так, использование современных видеоадаптеров с графическим процессором и поддержкой средств OpenGL позволяет повысить скорость построения трехмерных графиков в несколько раз и добиться большей их выразительности.

6.5.5. Построение графиков функций трех переменных

Графики сечений функций трех переменных строит команда `slice` (в переводе – «ломтик»). Она используется в следующих формах.

- `slice(X,Y,Z,V,Sx,Sy,Sz)` строит плоские сечения объемной фигуры `V` в направлении осей *x*, *y*, *z* с позициями, задаваемыми векторами `Sx`, `Sy`, `Sz`.

Массивы X, Y, Z задают координаты для V и должны быть монотонными и трехмерными (как возвращаемые функцией meshgrid) с размером M×N×P. Цвет точек сечений определяется трехмерной интерполяцией в объемной фигуре V.

- slice(X,Y,Z,V,XI,YI,ZI) строит сечения объемной фигуры V по поверхности, определенной массивами XI, YI, ZI.
- slice(V,Sx,Sy,Sz) или slice(V,XI,YI,ZI) – подразумевается X=1:N, Y=1:M, Z=1:P.
- slice(…,'method') – при построении задается метод интерполяции, который может быть одним из следующих: 'linear', 'cubic' или 'nearest'. По умолчанию используется линейная интерполяция – 'linear'.
- H=slice(…) строит сечение и возвращает дескриптор объекта класса surface.

```
% Программа построения графика функции трех переменных
[x,y,z] = meshgrid(-2:.2:2, -2:.25:2, -2:.16:2);
v = sin(x) .* exp(-x.^2 – y.^2 – z.^2);
slice(x,y,z,v,[-1.2 .8 2],2,[-2 -.2])
```

График, который строит эта программа, показан на рис. 6.30.

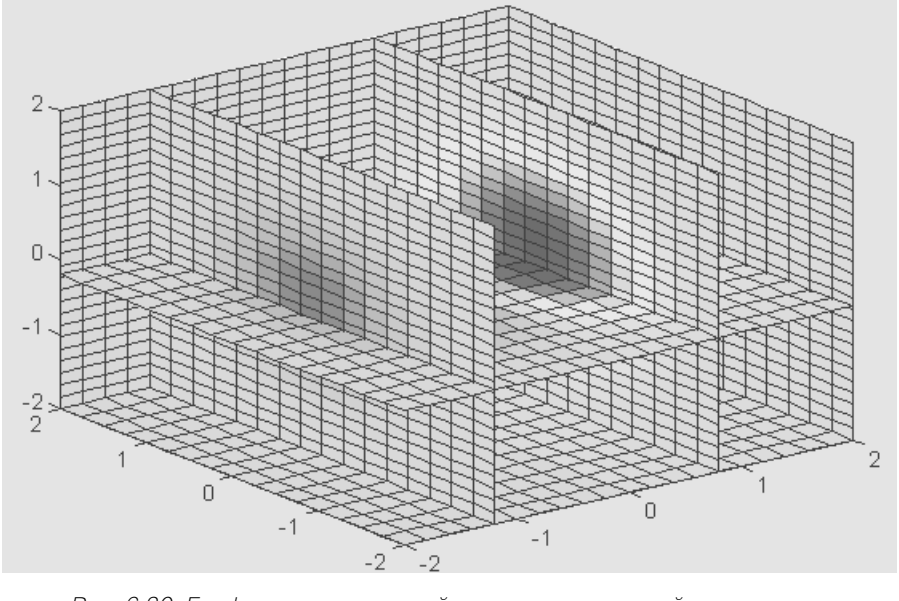

Рис. 6.30. График, показывающий сечения трехмерной поверхности

6.5.6. График трехмерной слоеной поверхности

Иногда бывают полезны графики трехмерных слоеных поверхностей, как бы состоящие из тонких пластинок – слоев. Такие поверхности строит функция `waterfall` (водопад):

- `waterfall(…)` строит поверхность как команда `mesh(…)`, но без показа ребер сетки. При ориентации графика относительно столбцов следует использовать запись `waterfall(Z')` или `waterfall(X',Y',Z')`. Пример:

```
% Программа построения графика слоенной поверхности
[X,Y]=meshgrid([-3:0.1:3]);
Z=sin(X)./(X.^2+Y.^2+0.3);
waterfall(X,Y,Z);
colormap(gray);
shading interp
```

Соответствующий график показан на рис. 6.31.

6.5.7. Трехмерные контурные графики

Трехмерный контурный график представляет собой расположенные в пространстве линии равного уровня, полученные при расслоении трехмерной фигуры рядом секущих плоскостей, расположенных параллельно опорной плоскости фигу-

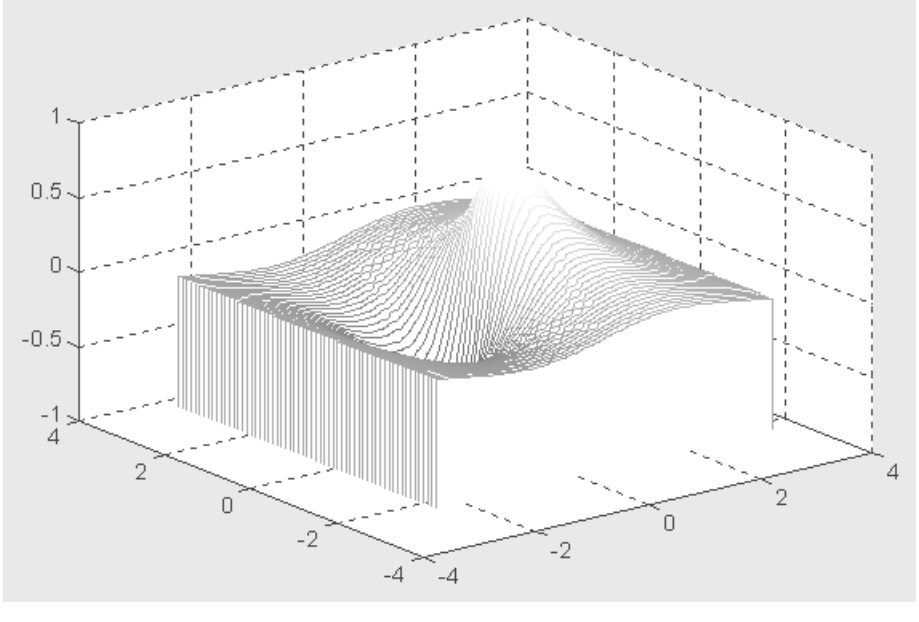

Рис. 6.31. Трехмерная слоеная поверхность

ры. При этом, в отличие от двумерного контурного графика, линии равного уровня отображаются в аксонометрии. Для получения трехмерных контурных графиков используется команда `contour3`:

- `contour3(...)` имеет синтаксис, аналогичный команде `contour(...)`, но строит линии равного уровня в аксонометрии с использованием функциональной окраски (окраска меняется вдоль оси `Z`).

Полезные частные формы записи этой команды:

- `contour3(Z)` строит контурные линии для поверхности, заданной массивом `Z`, без учета диапазона изменения x и y;
- `contour3(Z,n)` строит то же, что предыдущая команда, но с использованием `n` секущих плоскостей (по умолчанию `n=10`);
- `contour3(X,Y,Z)` строит контурные линии для поверхности, заданной массивом `Z`, с учетом изменения x и y. Двумерные массивы `X` и `Y` создаются с помощью функции `meshgrid`;
- `contour3(X,Y,Z,n)` строит то же, что предыдущая команда, но с использованием `n` секущих плоскостей.

Пример применения команды `contour3`:

```
>> contour3(peaks,20); colormap(gray)
```

Соответствующий данному примеру график представлен на рис. 6.32. В данном случае задано построение 20 линий уровня.

С командой `contour3` связаны следующие одноименные функции (не выполняющие графических построений).

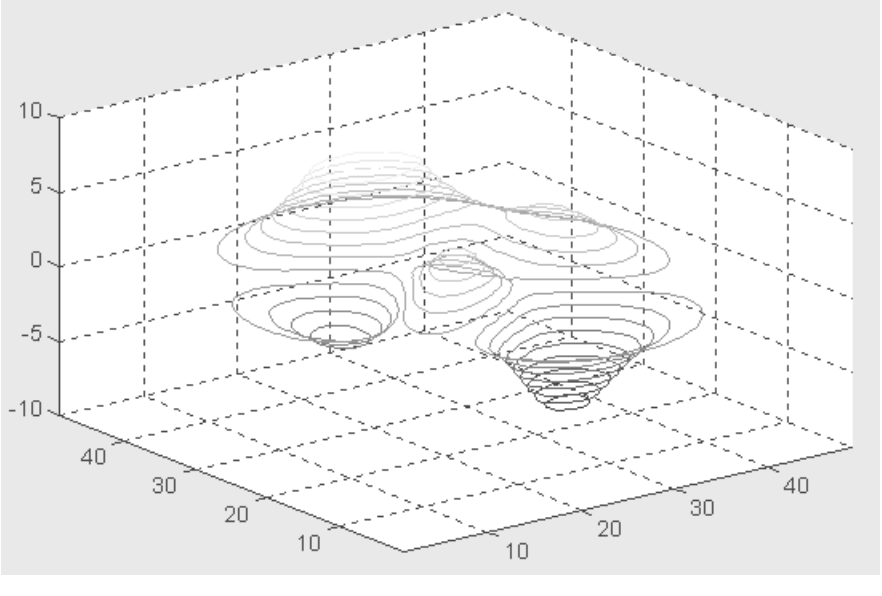

Рис. 6.32. Трехмерный контурный график для функции peaks

- `C=contour3(…)` возвращает матрицу описания контурных линий `C` для использования командой `clabel`.
- `[C,H]=contour3(…)` возвращает массив `C` и вектор-столбец `H` дескрипторов объектов `path` для каждой линии уровня. Свойство `UserData` каждого объекта содержит значение высоты для соответствующего контура.

6.6. Текстовое оформление графиков

6.6.1. Установка титульной надписи

После того как график уже построен, MATLAB позволяет выполнить его форматирование или оформление в нужном виде. Соответствующие этому средства описаны ниже. Так, для установки над графиком титульной надписи используется следующая команда:

- `title('string')` установка на двумерных и трехмерных графиках титульной надписи, заданной строковой константой `'string'`.

Пример применения этой команды будет дан в следующем разделе.

6.6.2. Установка осевых надписей

Для установки надписей возле осей *x*, *y* и *z* используются следующие команды:

```
xlabel('String')
ylabel('String')
zlabel('String')
```

Соответствующая надпись задается символьной константой или переменной `'String'`. Пример установки титульной надписи и надписей по осям графиков приводится ниже:

```
% Программа построения графика поверхности
% с текстовым оформлением
[X,Y]=meshgrid([-3:0.1:3]);
Z=sin(X)./(X.^2+Y.^2+0.3);
surfl(X,Y,Z); colorbar
colormap(gray); shading interp
xlabel('Axis X'); ylabel('Axis Y')
zlabel('Axis Z'); title('Surface graphic')
```

Построенный в этом примере график трехмерной поверхности показан на рис. 6.33.

Сравните его с графиком, показанным на рис. 6.29. Надписи делают рисунок более наглядным.

6.6.3. Ввод текста в любое место графика

Часто возникает необходимость добавления текста в определенное место графика, например для обозначения той или иной кривой графика. Для этого используется команда `text`:

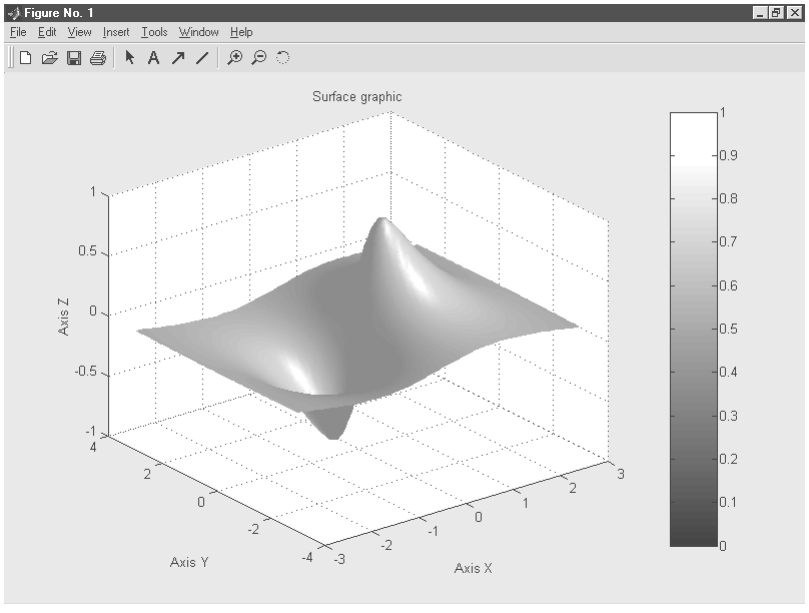

Рис. 6.33. График трехмерной поверхности с титульной надписью и надписями по координатным осям

- `text(X,Y,'string')` добавляет в двумерный график текст, заданный строковой константой `'string'`, так что начало текста расположено в точке с координатами (*X*,*Y*). Если `X` и `Y` заданы как одномерные массивы, то надпись помещается во все позиции `[x(i),y(i)]`;
- `text(X,Y,Z,'string')` добавляет в трехмерный график текст, заданный строковой константой `'string'`, так что начало текста расположено в позиции, заданной координатами `X`, `Y` и `Z`.

В приведенном ниже примере надпись «График функции sin(x^3)» размещается под кривой графика в позиции (–4, 0.7):

```
>> x=-10:0.1:10; plot(x,sin(x).^3)
>> text(-4,0.7,'Graphic sin(x)^3')
```

График функции с надписью у кривой показан на рис. 6.34.

Математически правильной записью была бы $\sin^3 x$. Попробуйте ввести самостоятельно

```
>> x=-10:0.1:10;
>> plot(x,sin(x).^3)
>> text(-4,0.7,'Graphic (sin(x))^3)')
```

Функция `h=text(…)` возвращает вектор-столбец `h` дескрипторов объектов класса `text`, дочерних для объектов класса `axes`. Следующий пример вычисляет дескриптор `h`:

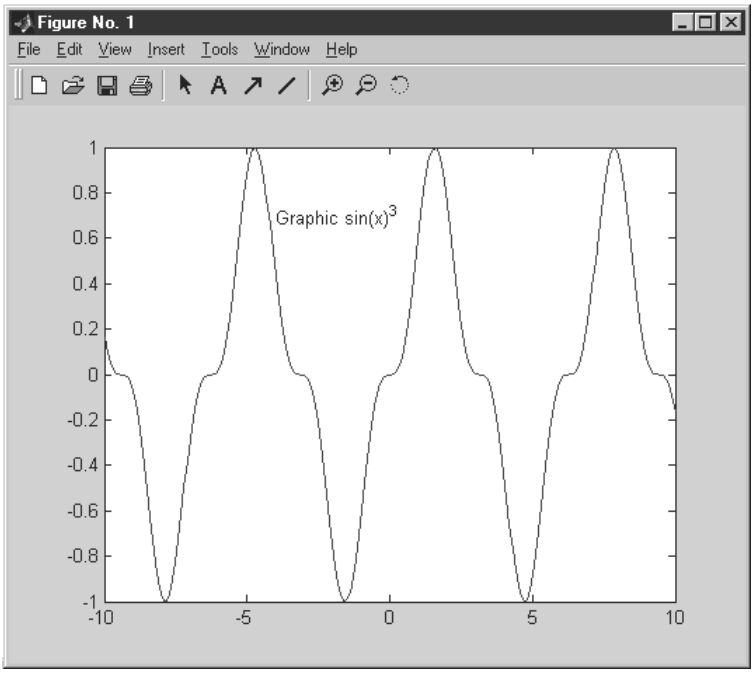

Рис. 6.34. Пример ввода надписи в поле графика функции

```
>> h=text(.25,.5,'\ite^{i\omega\tau} = cos(\omega\tau) + … i
sin(\omega\tau)')
h = 3.0022
```

и выводит в пустом графике математическую формулу в формате TeX вида:

$e^{j\omega t} = \cos(\omega t) + \sin(\omega t)$.

Пары координат `X,Y` (или тройки `X,Y,Z` для трехмерных графиков) могут сопровождаться парами «имя параметра/значение параметра» для задания дополнительных свойств текста. Пары координат `X,Y` (или тройки `X,Y,Z` для трехмерных графиков) могут быть полностью опущены, при этом все свойства, в том числе и позиция текста, задаются с помощью пар «имя параметра/значение параметра», определенных по умолчанию.

Используйте функцию `get(H)`, где `H` – дескриптор графического объекта (в нашем случае графического объекта класса `text`), чтобы просмотреть список свойств объекта и их текущие значения. Используйте `set(H)`, чтобы просмотреть список свойств графических объектов и их допустимых значений.

6.6.4. Позиционирование текста с помощью мыши

Очень удобный способ ввода текста предоставляет команда `gtext`:

- `gtext('string')` задает выводимый на график текст в виде строковой константы `'string'` и выводит на график перемещаемый мышью маркер в виде крестика. Установив маркер в нужное место, достаточно щелкнуть любой кнопкой мыши для вывода текста;
- `gtext(C)` позволяет аналогичным образом разместить многострочную надпись из массива строковых переменных `C`.

Пример применения команды `gtext`:

```
>> x=-15:0.1:15; plot(x, sin(x).^3)
>> gtext('Function sin(x)^3')
```

При исполнении этого примера вначале можно увидеть построение графика функции с большим крестом, перемещаемым мышью (рис. 6.35).

Установив перекрестие в нужное место графика, достаточно нажать любую клавишу или любую кнопку мыши, и на этом месте появится надпись (рис. 6.36).

Высокая точность позиционирования надписи и быстрота процесса делают данный способ нанесения надписей на графики одним из наиболее удобных.

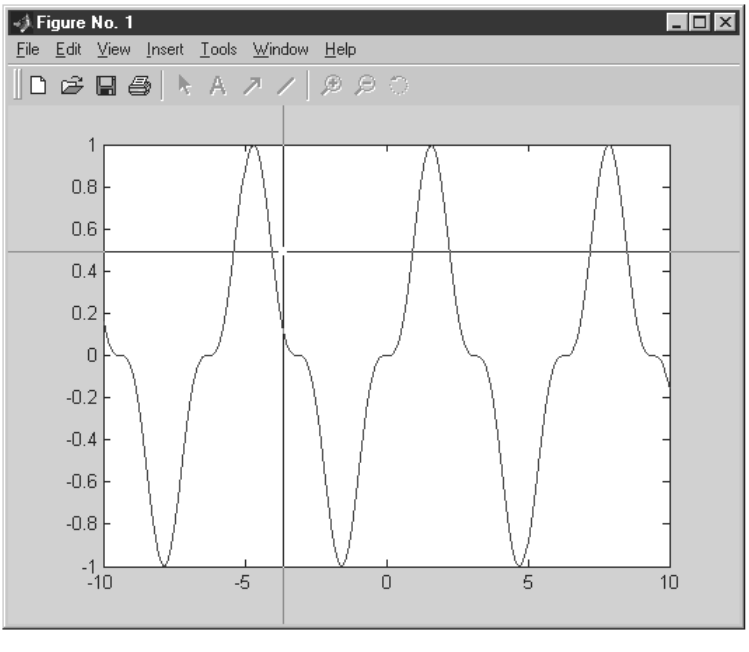

Рис. 6.35. График функции с крестообразным маркером, перемещаемым мышью

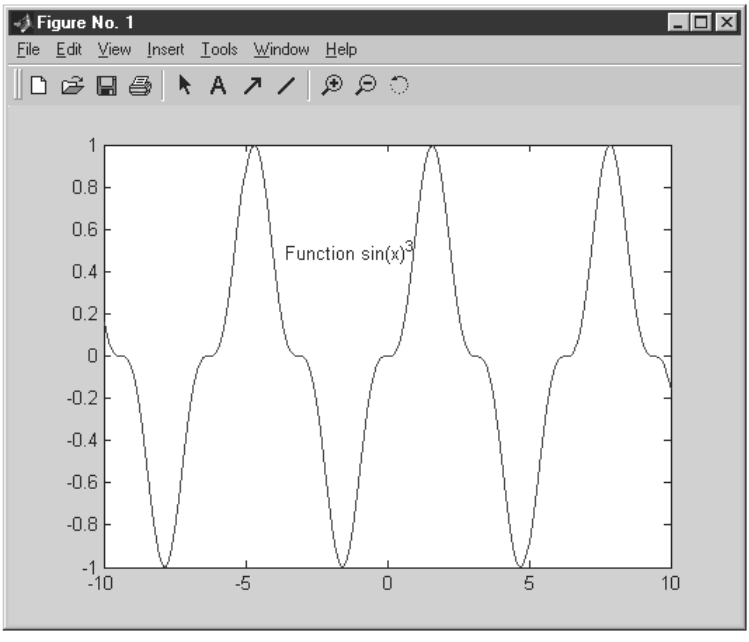

Рис. 6.36. График функции с надписью, установленной с помощью мыши

6.7. Форматирование графиков

6.7.1. Вывод пояснений и легенды

Пояснение в виде отрезков линий со справочными надписями, размещаемое внутри графика или около него, называется *легендой*. Для создания легенды используются различные варианты команды `legend`:

- `legend(string1,string2,string3,…)` добавляет к текущему графику легенду в виде строк, указанных в списке параметров;
- `legend(H,string1,string2,string3,…)` помещает легенду на график, содержащий объекты с дескрипторами `H`, используя заданные строки как метки для соответствующих дескрипторов;
- `legend(AX,…)` помещает легенду в осях (объект класса `axes`) с дескриптором `AX`;
- `legend(M)` размещает легенду, используя данные из строковой матрицы `M`;
- `legend OFF` устраняет ранее выведенную легенду;
- `legend` перерисовывает текущую легенду, если таковая имеется;

- `legend(legendhandle)` перерисовывает легенду, указанную дескриптором `legendhandle`;
- `legend(…,Pos)` помещает легенду в точно определенное место, специфицированное параметром `Pos`:
  - `Pos=0` – лучшее место, выбираемое автоматически;
  - `Pos=1` – верхний правый угол;
  - `Pos=2` – верхний левый угол;
  - `Pos=3` – нижний левый угол;
  - `Pos=4` – нижний правый угол;
  - `Pos=-1` – справа от графика.

Чтобы перенести легенду, установите на нее курсор, нажмите левую кнопку мыши и перетащите легенду в необходимую позицию.

- `[legh,objh]=legend(…)` – эта функция возвращает дескриптор объекта для легенды (`legh`) и матрицу `objh`, содержащую дескрипторы объектов, из которых легенда состоит.

Команда `legend` может использоваться с двумерной и трехмерной графикой и со специальной графикой – столбцовыми и круговыми диаграммами и т. д. Двойным щелчком можно вывести легенду на редактирование.

Программа, приведенная ниже, строит график трех функций с легендой, размещенной в поле графика:

```
% Программа построения графика трех функций
% с выводом их обозначений — легендой
x=-2*pi:0.1*pi:2*pi;
y1=sin(x); y2=sin(x).^2; y3=sin(x).^3;
plot(x,y1,'-m',x,y2,'-.+r',x,y3,'-ok')
legend('Function 1','Function 2','Function 3');
```

Полученный график представлен на рис. 6.37.

Незначительная модификация команды `legend` (применение дополнительного параметра `-1`) позволяет построить график трех функций с легендой вне поля графика. Это иллюстрирует следующая программа:

```
% Программа построения графика трех функций
% с выводом легенды вне поля графика
x=-2*pi:0.1*pi:2*pi;
y1=sin(x); y2=sin(x).^2; y3=sin(x).^3;
plot(x,y1,'-m',x,y2,'-.+r',x,y3,'-ok')
legend('Function 1','Function 2','Function 3',-1);
```

Соответствующий график показан на рис. 6.38.

В данном случае недостатком можно считать сокращение полезной площади самого графика. Остальные варианты расположения легенды пользователю предлагается изучить самостоятельно. Следует отметить, что применение легенды придает графикам более осмысленный и профессиональный вид. При необходимости легенду можно переместить мышью в подходящее место графика.

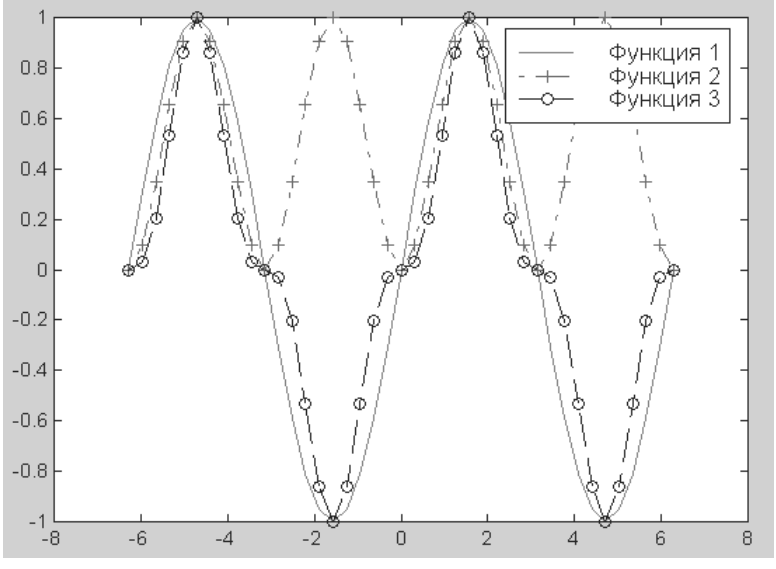

Рис. 6.37. График трех функций с легендой в поле графика

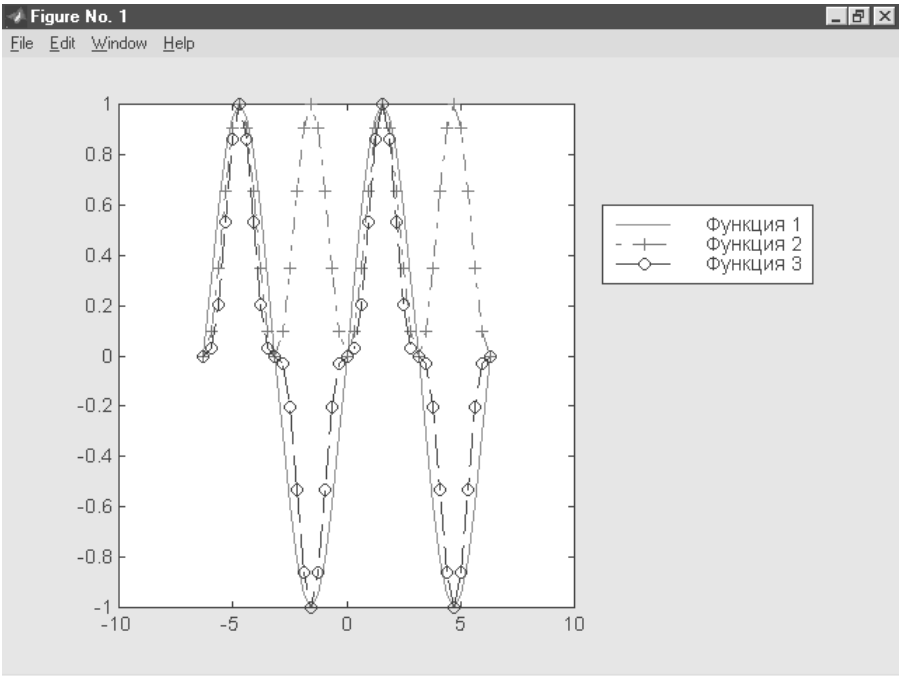

Рис. 6.38. График трех функций с легендой, расположенной вне поля графика

6.7.2. Маркировка линий уровня на контурных графиках

К сожалению, контурные графики плохо приспособлены для количественных оценок, если их линии не маркированы. В качестве маркеров используются крестики, рядом с которыми располагаются значения высот. Для маркировки контурных графиков используются команды группы `clabel`:

- `clabel(CS,H)` маркирует контурный график с данными в контурной матрице `CS` и дескрипторами объектов, заданными в массиве `H`. Метки вставляются в разрывы контурных линий и ориентируются в соответствии с направлением линий;
- `clabel(CS,H,V)` маркируются только те уровни, которые указаны в векторе `V`. По умолчанию маркируются все контуры. Позиции меток располагаются случайным образом;
- `clabel(CS,H,'manual')` маркирует контурные графики с установкой положения маркеров с помощью мыши. Нажатие клавиши **Enter** или кнопки мыши завершает установку маркера. При отсутствии мыши для перехода от одной линии уровня к другой используется клавиша пробела, а для перемещения надписи используются клавиши перемещения курсора;
- `clabel(CS)`, `clabel(CS,V)` и `clabel(CS,'manual')` – дополнительные возможности маркировки контурных графиков. При отсутствии аргумента *h* метки не ориентируются вдоль линий контуров; точную позицию метки отмечает значок «плюс» (далее на рис. 6.39 показан именно этот вариант).

Пример применения команды `clabel` приводится ниже:

```
% Программа построения контурного графика поверхности
% с маркированными линиями уровня
[X,Y]=meshgrid([-3:0.1:3]);
Z=sin(X)./(X.^2+Y.^2+0.3); C=contour(X,Y,Z,10);
colormap(gray); clabel(C)
```

Рисунок 6.39 показывает построение контурного графика с маркированными линиями уровня, полученного при исполнении приведенной программы.

Функция `H=clabel(...)` маркирует график и возвращает дескрипторы создаваемых при маркировке объектов класса `TEXT` (и, возможно, `LINE`).

6.7.3. Управление свойствами осей графиков

Обычно графики выводятся в режиме автоматического масштабирования. Следующие команды класса `axis` меняют эту ситуацию:

- `axis([XMIN XMAX YMIN YMAX])` – установка диапазонов координат по осям *x* и *y* для текущего двумерного графика;

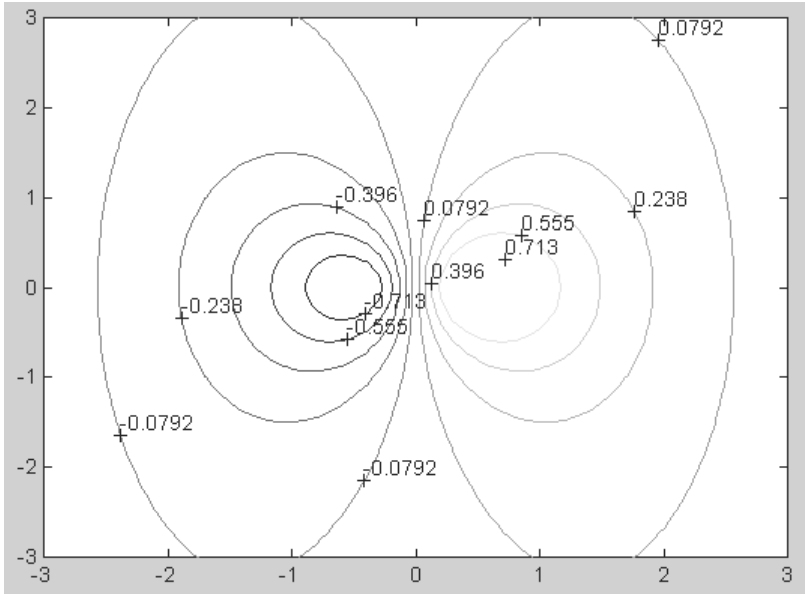

Рис. 6.39. Контурный график с маркированными линиями уровня

- `axis([XMIN XMAX YMIN YMAX ZMIN ZMAX])` – установка диапазонов координат по осям *x*, *y* и *z* текущего трехмерного графика;
- `axis auto` – установка параметров осей по умолчанию;
- `axis manual` «замораживает» масштабирование в текущем состоянии, чтобы при использовании команды `hold on` следующие графики использовали те же параметры осей;
- `axis tight` устанавливает диапазоны координат по осям в соответствии с диапазонами изменения данных;
- `axis ij` задает «матричную» прямоугольную систему координат с началом координат в левом верхнем углу, ось *i* – вертикальная, размечаемая сверху вниз, ось *j* – горизонтальная и размечается слева направо;
- `axis xy` устанавливает декартову систему координат с горизонтальной осью *x*, размечаемой слева направо, и вертикальной осью *y*, размечаемой снизу вверх (начало координат размещается в нижнем левом углу);
- `axis equal` включает масштаб с одинаковым расстоянием между метками по осям *x*, *y* и *z*;
- `axis image` устанавливает масштаб, при котором пиксели изображения становятся квадратами;
- `axis square` устанавливает текущие оси в виде квадрата (или куба в трехмерном случае) с одинаковым расстоянием между метками и одинаковой длиной осей;
- `axis normal` восстанавливает масштаб, отменяя установки `axis equal` и `axis square`;

- `axis vis3d` «замораживает» пропорции осей для возможности поворота трехмерных объектов;
- `axis off` убирает с осей их обозначения и маркеры;
- `axis on` восстанавливает ранее введенные обозначения осей и маркеры;
- `V=axis` возвращает вектор-строку, содержащую коэффициенты масштабирования для текущего графика. Если текущий график двумерный, то вектор имеет 4 компонента, если трехмерный – 6 компонентов.

Следующий пример иллюстрирует применение команды `axis` при построении двумерного графика функции одной переменной:

```
>> x=-5:0.1:5; plot(x,sin(x)); axis([-10 10 -1.5 1.5])
```

На рис. 6.40 показано изображение, которое строится в этом примере.

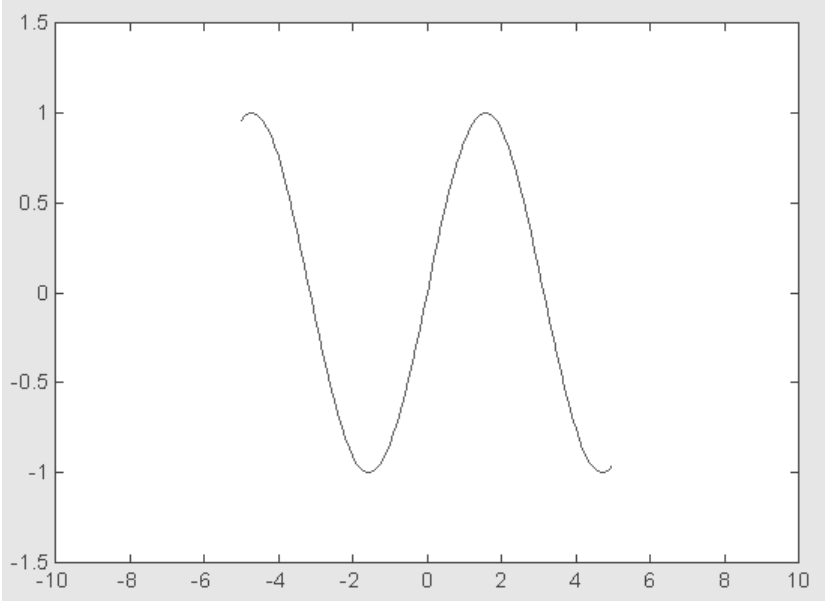

Рис. 6.40. Пример задания масштаба осей двумерного графика

Обратите внимание, что теперь масштабы осей заданы командой `axis`, а не диапазоном изменения значений x и y.

6.7.4. Включение и выключение сетки

В математической, физической и иной литературе при построении графиков в дополнение к разметке осей часто используют масштабную сетку. Команды `grid` позволяют задавать построение сетки или отменять это построение:

- `grid on` добавляет сетку к текущему графику;

- `grid off` отключает сетку;
- `grid` последовательно производит включение и отключение сетки.

Команды `grid` устанавливают свойства объектов `XGrid`, `Ygrid` и `Zgrid` для текущих осей. Ниже приведен пример из предшествующего раздела с добавлением в него команды `grid`:

```
>> x=-5:0.1:5; plot(x,sin(x));
>> axis([-10 10 -1.5 1.5]); grid on
```

Построенный график показан на рис. 6.41.

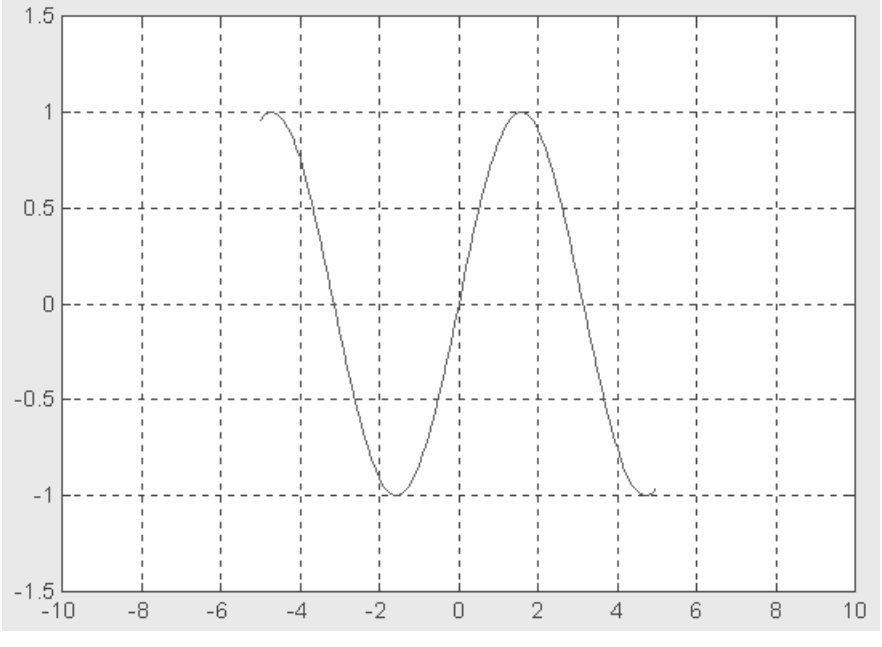

Рис. 6.41. График синусоиды с сеткой разметки

Сравните этот график с графиком на рис. 6.40, на котором сетка отсутствует. Нетрудно заметить, что наличие сетки облегчает количественную оценку координат точек графика.

6.7.5. Наложение графиков друг на друга

Во многих случаях желательно построение многих наложенных друг на друга графиков в одном и том же окне. Для этого служит команда продолжения графических построений `hold`. Она используется в следующих формах:

- `hold on` обеспечивает продолжение вывода графиков в текущее окно, что позволяет добавлять последующие графики к уже существующим;

- `hold off` отменяет режим продолжения графических построений;
- `hold` работает как переключатель, последовательно включая режим продолжения графических построений и отменяя его.

Команда `hold on` устанавливает значение `add` для свойства `NextPlot` объектов `figure` и `axes`, а `hold off` устанавливает для этого свойства значение `replace`. Рекомендуется также ознакомиться с командами `ishold`, `newplot`, `figure` и `axes`.

Приведенный ниже пример показывает, как с помощью команды `hold on` на график синусоиды накладываются еще три графика параметрически заданных функций:

```
>> x=-5:0.1:5; plot(x,sin(x)); hold on
>> plot(sin(x),cos(x)); plot(2*sin(x),cos(x))
>> plot(4*sin(x),cos(x)); hold off
```

Графики построенных функций показаны на рис. 6.42.

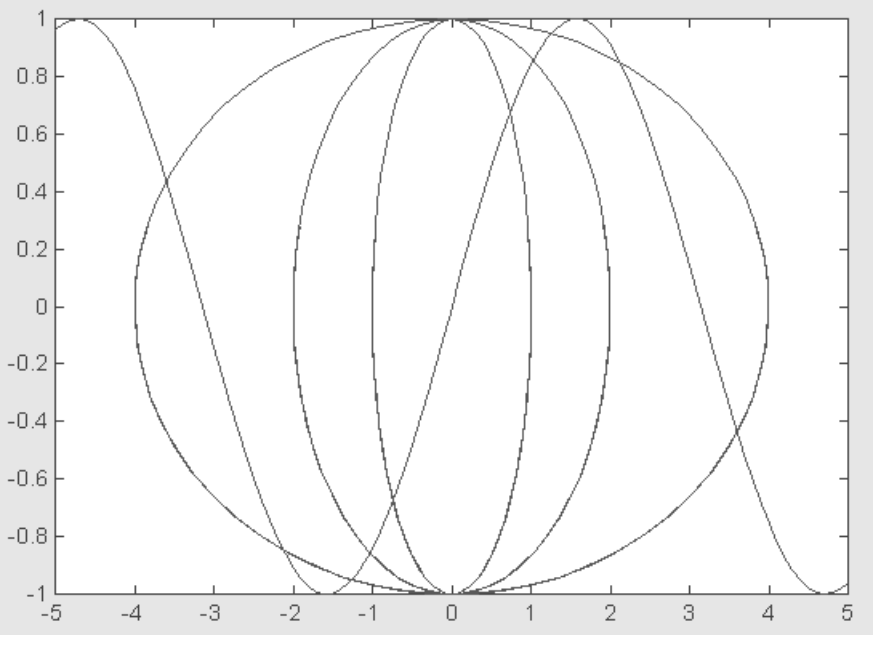

Рис. 6.42. Графики синусоиды и трех параметрических функций в одном окне

В конце приведенного фрагмента программы команда `hold off` отключает режим добавления графиков к ранее построенным графикам.

6.7.6. Разбиение графического окна

Бывает, что в одном окне надо расположить несколько координатных осей с различными графиками без наложения их друг на друга. Для этого используются команды `subplot`, применяемые перед построением графиков:

- `subplot` создает новые объекты класса `axes` (подокна);
- `subplot(m,n,p)` или `subplot(mnp)` разбивает графическое окно на $m \times n$ подокон, при этом `m` – число подокон по горизонтали, `n` – число подокон по вертикали, а `p` – номер подокна, в которое будет выводиться текущий график (подокна отсчитываются последовательно по строкам);
- `subplot(H)`, где `H` – дескриптор для объекта `axes`, дает альтернативный способ задания подокна для текущего графика;
- `subplot('position',[left bottom width height])` создает подокно с заданными нормализованными координатами (в пределах от `0.0` до `1.0`);
- `subplot(111)` и `clf reset` удаляют все подокна и возвращают графическое окно в обычное состояние.

Следующая программа иллюстрирует применение команды `subplot`:

```
% Программа построения четырех графиков в разных подокнах
x=-5:0.1:5;
subplot(2,2,1),plot(x,sin(x))
subplot(2,2,2),plot(sin(5*x),cos(2*x+0.2))
subplot(2,2,3),contour(peaks)
subplot(2,2,4),surf(peaks)
```

В этом примере при пуске программы последовательно строятся четыре графика различного типа, размещаемых в разных подокнах (рис. 6.43).

Следует отметить, что для всех графиков возможна индивидуальная установка дополнительных объектов, например титульных надписей, надписей по осям и т. д.

6.7.7. Изменение масштаба графика

Для изменения масштаба двумерных графиков используются команды класса `zoom`:

- `zoom` переключает состояние режима интерактивного изменения масштаба для текущего графика;
- `zoom(FACTOR)` устанавливает масштаб в соответствии с коэффициентом `FACTOR`;
- `zoom on` включает режим интерактивного изменения масштаба для текущего графика;
- `zoom off` выключает режим интерактивного изменения масштаба для текущего графика;
- `zoom out` обеспечивает полный просмотр, то есть устанавливает стандартный масштаб графика;
- `zoom xon` или `zoom yon` включает режим изменения масштаба только по оси x или по оси y;

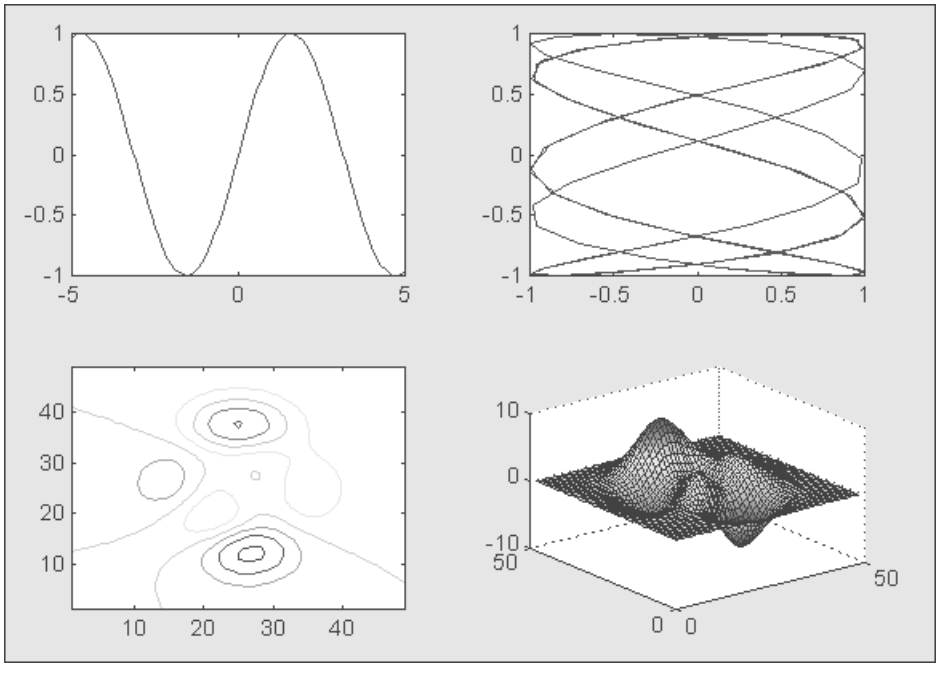

Рис. 6.43. Четыре графика различного типа, размещенных в подокнах одного окна

- `zoom reset` запоминает текущий масштаб в качестве масштаба по умолчанию для данного графика;
- `zoom(FIG,OPTION)` применяется к графику, заданному дескриптором `FIG`, при этом `OPTION` может быть любым из перечисленных выше аргументов.

Команда `zoom` позволяет управлять масштабированием графика с помощью мыши. Для этого надо подвести курсор мыши к интересующей вас области рисунка. Если команда `zoom` включена (`on`), то нажатие левой кнопки увеличивает масштаб вдвое, а правой – уменьшает вдвое. При нажатой левой кнопке мыши можно выделить пунктирным черным прямоугольником нужный участок графика – при отпускании кнопки он появится в увеличенном виде и в том масштабе, который соответствует выделяющему прямоугольнику.

Рассмотрим работу команды `zoom` на следующем примере:

```
>> x=-5:0.01:5; plot(x,sin(x.^5)./(x.^5+eps)); zoom on
```

Рисунок 6.44 показывает график функции данного примера в режиме выделения его участка с помощью мыши.

После прекращения манипуляций левой кнопкой мыши график примет вид, показанный на рис. 6.45. Теперь в полный размер графического окна будет развернуто изображение, попавшее в выделяющий прямоугольник.

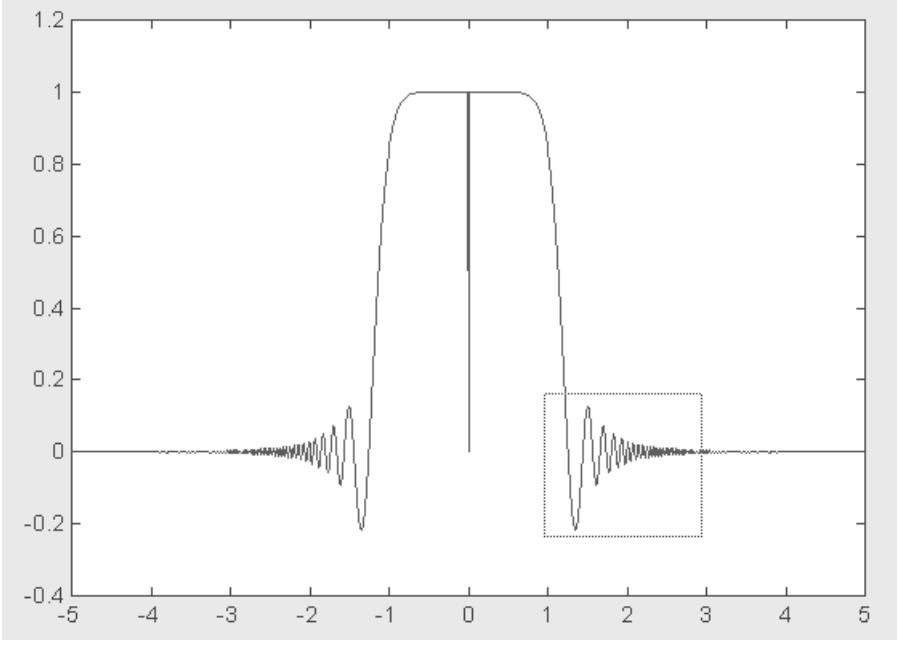

Рис. 6.44. Выделение части графика мышью при использовании команды zoom

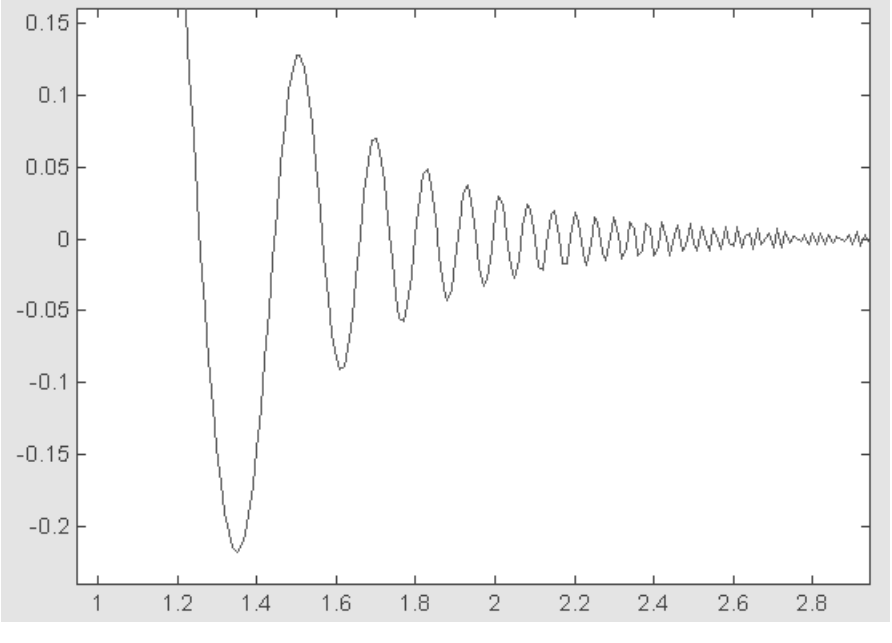

Рис. 6.45. График выделенного участка

Команда zoom, таким образом, выполняет функцию «лупы», позволяющей наблюдать в увеличенном виде отдельные фрагменты сложных графиков. Однако следует учитывать, что для наблюдения фрагментов графиков при высоком увеличении они должны быть заданы большим количеством точек. Иначе вид отдельных фрагментов и тем более особых точек (в нашем случае это точка при x вблизи нуля) будет существенно отличаться от истинного.

6.8. Цветовая окраска графиков

6.8.1. Установка палитры цветов

Поскольку графика MATLAB обеспечивает получение цветных изображений, в ней есть ряд команд для управления цветом и различными световыми эффектами. Среди них важное место занимает установка палитры цветов. Палитра цветов RGB задается матрицей MAP из трех столбцов, определяющих значения интенсивности красного (red), зеленого (green) и синего (blue) цветов. Их интенсивность задается в относительных единицах от 0.0 до 1.0. Например, [0 0 0] задает черный цвет, [1 1 1] – белый цвет, [0 0 1] – синий цвет. При изменении интенсивности цветов в указанных пределах возможно задание любого цвета. Таким образом, цвет соответствует общепринятому формату RGB.

Для установки палитры цветов служит команда colormap, записываемая в следующих формах:

- colormap('default') устанавливает палитру по умолчанию, при которой распределение цветов соответствует радуге;
- colormap(MAP) устанавливает палитру RGB, заданную матрицей MAP;
- C=colormap – функция возвращает матрицу текущей палитры цветов C.

m-файл с именем **colormap** устанавливает свойства цветов для текущего графика.

Команда help graph3d наряду с прочим выводит полный список характерных палитр, используемых графической системой MATLAB:

- hsv – цвета радуги;
- hot – чередование черного, красного, желтого и белого цветов;
- gray – линейная палитра в оттенках серого цвета;
- bone – серые цвета с оттенком синего;
- copper – линейная палитра с оттенками меди;
- pink – розовые цвета с оттенками пастели;
- white – палитра белого цвета;
- flag – чередование красного, белого, синего и черного цветов;
- lines – палитра с чередованием цветов линий;
- colorcube – расширенная палитра RGB;
- jet – разновидность палитры HSV;
- prism – призматическая палитра цветов;
- cool – оттенки голубого и фиолетового цветов;
- autumn – оттенки красного и желтого цветов;