Андриевский А.Б., Андриевский Б.Р.,
Капитонов А.А., Фрадков А.Л.

РЕШЕНИЕ ИНЖЕНЕРНЫХ ЗАДАЧ В SCILAB

Санкт-Петербург
2013

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

КАФЕДРА СИСТЕМ УПРАВЛЕНИЯ И ИНФОРМАТИКИ

Андриевский А.Б., Андриевский Б.Р., Капитонов А.А., Фрадков А.Л.

РЕШЕНИЕ ИНЖЕНЕРНЫХ ЗАДАЧ В SCILAB

Учебное пособие

Санкт-Петербург

2013

УДК 681.51, 681.53, 681.58

Андриевский А.Б., Андриевский Б.Р., Капитонов А.А., Фрадков А.Л. Решение инженерных задач в среде Scilab. Учебное пособие.— СПб.: НИУ ИТМО, 2013. — 97 с.

Содержатся основные сведения и практические рекомендации по работе с пакетом Scilab, предназначенным для выполнения широкого круга инженерных и научных расчетов. Язык программирования Scilab схож с языком системы MATLAB, но представленный пакет является свободно распространяемым (некоммерческим) продуктом. Scilab позволяет работать с элементарными и специальными функциями, имеет мощные средства для работы с матрицами, полиномами, решения дифференциальных уравнений, оптимизации, работы с графиками. В состав пакета также входит Xcos — инструмент моделирования с графическим интерфейсом (аналог Simulink в пакете MATLAB).

Илл. 47, список литературы — 10 наим.

Одобрено на заседании кафедры СУиИ, протокол № 5 от 27.06.2013
Одобрено Ученым советом факультета КТиУ, протокол № 9 от 08.10.2013

В 2009 году Университет стал победителем многоэтапного конкурса, в результате которого определены 12 ведущих университетов России, которым присвоена категория “Национальный исследовательский университет”. Министерством образования и науки Российской Федерации была утверждена программа его развития на 2009-2018 годы. В 2011 году Университет получил наименование “Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики”.

Оглавление

Глава 1

Основные конструкции языка

1.1 Константы

1.1.1 Вещественные числа

В настоящем разделе дано беглое описание основных конструкций языка Scilab. Более полные сведения можно найти в [1–5].

Scilab поддерживает различные способы записи вещественных чисел. Например, допустимы записи: `4`, `-5.2`, `32e3`, `7.8e-5`, `-12E-10`, `-12D-10`. Разделителем между мантиссой и порядком при вводе могут служить символы `e`, `E`, `d`, `D`. При выводе на экран Scilab использует символ `D`. Заметим, что разделителем между целой и дробной частями числа, как это принято в программировании, служит символ «точка», а не «запятая». Запись `-5,2` соответствует двум выражениям (`-5` и `2`).

В среде Scilab не делается различия между целыми и вещественными числами.

1.1.2 Мнимые числа

Для задания мнимых (комплексных) чисел используется предопределенная константа `%i`, обозначающая *мнимую единицу* $i = \sqrt{-1}$. Например, записи `43*%i`, `5-3*%i` соответствуют комплексным числам $43i$, $5 - 3i$.

Для извлечения вещественной и мнимой части числа используются функции `real()` и `imag()`. Так, если набрать в командной строке выражение `imag(5-3*%i)`, то получим результат `ans = -3`.

Это значит, что переменной с резервированным именем `ans` (от англ. *answer* – ответ) присвоено значение мнимой части комплексного числа $5 - 3i$.

1.1.3 Стандартные константы

Часто используемые величины предопределены в Scilab как следующие константы:
`%e` – число e (основание натурального логарифма);
`%pi` – число π;
`%inf` – «бесконечность»;
`%nan` – «не число» («*not a number*»);
`%t` или `%T` – логическая единица, «истина» («*true*»);
`%f` или `%F` – логический нуль, «ложь» («*false*»);
`%z` – полином с единственным нулевым корнем и аргументом z;
`%s` – полином с единственным нулевым корнем и аргументом s.

1.2 Основные типы данных

1.2.1 Скалярные объекты

Оператором присваивания
```
a=1
```
вводится *вещественная переменная* a, которой присваивается значение 1.

Оператор
```
b=%t
```
вводит *переменную логического типа*, которой устанавливается значение `%t` («истина»).

Строка символов («*string*») вводится с помощью одиночных или двойных апострофов, например
```
s='This is a string'
```

На языке Scilab можно описывать данные типа «*многочлен*». Выполним оператор
```
p=poly(3,'z')
```
Его результатом будет многочлен первой степени от агрумента z, имеющий один корень 3:

```
p =
  - 3 + z
```

Встроенная функция `poly` имеет два аргумента. Первым аргументом является массив корней многочлена (о задании массивов см. ниже, п. 1.2.2). Второй аргумент – строковая константа из одного символа. При дальнейших действиях этот символ в программе выступает в качестве аргумента многочлена.

Многочлен можно задавать и его коэффициентами. Для этого сначала определим следующий многочлен, имеющий один нулевой корень:

```
z=poly(0,'z')
```

Фактически мы определили независимую переменную z, через которую обычными математическаими операциями можно задавать другие многочлены. Например, оператор

```
p=1+3*z+4.5*z^2
```

приводит к результату

```
p =
                  2
    1 + 3z + 4.5z
```

1.2.2 Массивы

Более сложными объектами языка являются *массивы* и *списки*.

С точки зрения синтаксиса такие математические объекты, как векторы и матрицы представляются массивами с одним или двумя измерениями (соответственно). Термины «*массив*» и «*матрица*» (или «*вектор*») будут далее использоваться как синонимы и употребляться в зависимости от контекста. Язык Scilab позволяет работать и с массивами, имеющими более двух измерений, которые нельзя трактовать как матрицы.

Для задания массивов через значения их элементов, используются квадратные скобки. Например массив `X`, состоящий из шести элементов задается оператором `X=[1 3 4 5 3 7]`

Введенный так массив `X` соответствует вектор-строке

$X = [1, 2, 4, 5, 3, 7] \in \mathbb{R}^{1\times 6}$.

Обращение к элементам массива производится через круглые скобки. Например, для доступа ко второму элементу следует ввести команду `x=X(2).` В результате получим `x`= 3.

В отличие от массива *список* может содержать элементы различных типов.

1.2.3 Списки

Список – это упорядоченная последовательность элементов, каждый из которых, в свою очередь, может быть либо списком, либо *атомарным неделимым элементом*. Список может использоваться для группировки разнотипных объектов в единый объект данных. Для создания списка используется оператор `list`. Элементы списка могут быть различных типов. Например:

```
L=list(a,-(1:5), Mp,['this','is';'a','list'])
```

получим следующий список:

```
L  =
       L(1)
    3.    5.    2.    6.    8.    3.    5.    8.
       L(2)
  - 1.  - 2.  - 3.  - 4.  - 5.
       L(3)
                  2
    1 + 3z + 4.5z     1 - z
                            2      3
    1                 z + 3z + 4.5z
       L(4)
!this  is    !
!            !
!a     list  !
```

Создадим теперь *типизированный список*, с помощью которого можно описывать новые абстрактные типы данных в Scilab. Название класса и его полей задаются с помощью вектора - строки в первом элементе типизированного списка. Например:

```
Lt=tlist(['mylist','color','position', 'weight'],
'blue',[0,1],10)
```

Первый аргумент функции `list` – вектор строк, первый элемент которго описывает сам список, а последующие определяют типы элементов списка.

```
 Lt =
       Lt(1)
!mylist  color  position  weight  !
       Lt(2)
 blue
       Lt(3)
    0.    1.
       Lt(4)
    10.
```

Таким образом, типу `'color'` будет соответствовать элемент `'blue'`. Обращение к элементу типизированного списка осуществляется следующим образом: `Lt('color').` Добавить к списку еще один элемент можно так: `L($+1)=B.`Таким оразом, мы добавили в конец списка матрицу B. Объединить два списка можно командой `NewL=lstcat(L,L1).`Этим мы сформировали новый список, который состоит из элементов списков L и $L1$.

1.3 Действия над матрицами

Для начала создадим вектор из восьми элементов. Для этого введем в командной строке оператор:

```
a = [3 5 2 6 8 3 5 8]
```

после нажатия клавиши `Enter` в командном окне появится ответ:

```
a =
    3.    5.    2.    6.    8.    3.    5.    8.
```

Для перехода к следующей строке при определении матрицы используется символ «`;`». Например, в результате выполнения оператора:

```
X = [2 6 -3 2; 6 7 9 8]
```

получим массив `X` размера (2×4):

```
X =
    2.    6.  - 3.    2.
    6.    7.    9.    8.
```

который соответствует матрице:

$$X = \begin{bmatrix} 2 & 6 & -3 & 2 \\ 6 & 7 & 9 & 8 \end{bmatrix} \in \mathbb{R}^{2\times 4}.$$

Элементы массивов должны быть одного общего типа, но не обязательно числового. Так, кроме массива вещественных чисел можно задать и *массив строк*:

```
St = ['this' 'is' 'a ' 'string' 'array']
```

Получим переменную:

```
St  =
!this  is  a   string  array  !
```

Тогда в ответ на оператор `whos`, вызывающий на экран список используемых системой переменных, получим:

```
St  string          1 by 5          120
```

Это означает, что двумерный массив `St` строкового типа имеет размерность 1 по первому измерению, размерность 5 по второму и занимает 120 байт памяти.

Можно задать и массив логических констант:

```
B = [%t %f],
```

или массив (матрицу) многочленов.

Для определения многочленной матрицы необходимо сначала задать составляющие ее многочлены. Зададим многочлен $P(z)$ вида $P(z) = 1 + 3z + 4.5z^2$:

```
P = 1+3*%z+4.5*%z^2;
```

Заметим, что символ «;» в конце оператора используется, чтобы результат его выполнения не выводился на дисплей.

Теперь создадим матрицу многочленов:

```
Mp = [P, 1-%z; 1, %z*P]
```

В командном окне увидим результат:

```
Mp =
                  2
    1 + 3z + 4.5z     1 - z
                            2      3
    1                 z + 3z + 4.5z
```

Для того чтобы узнать размер массива, используется оператор `size()`. Например, узнаем размер ранее введенного массива `A`. Результатом выполнения команды `size(A)` будет: `ans`= 2. 4. Это значит, что массив `A` имеет две строки и четыре столбца.

Рассмотрим теперь некоторые операции над матрицами. В результате выполнения команды `C=X+4` мы увидим

```
C =
    6.     10.    1.     6.
    10.    11.    13.    12.
```

Это значит, что каждый элемент матрицы `C` получен увеличением элементов матрицы `X` на 4.

Создадим теперь две матрицы размером 2×3:

```
A = [2 4 3; 7 5 1];
B = [5 2 8; -1 9 0];
```

Попробуем их перемножить:

```
D=A*B
```

Результатом будет сообщение об ошибке, так как операции матричного умножения в системе Scilab понимаются в векторно-матричном смысле:

```
       !--error 10
inconsistent multiplication
```

Для получения корректного ответа транспонируем одну из матриц. Операция транспонирования вещественных матриц задается символом «’» (апостроф) [1]:

```
C = X'
```

Получим результат:

```
C =
    2.    6.
    6.    7.
  - 3.    9.
    2.    8.
```

Выполним теперь операторы:

```
D=A'*B
```

и

```
E=A*B'
```

В первом случае получим:

```
D =
    3.     67.    16.
    15.    53.    32.
    14.    15.    24.
```

а во втором –

```
E =
    42.    34.
    53.    38.
```

Математически это значит, что в результате действий с матрицами $A, B \in \mathbb{R}^{2\times 3}$ получены матрицы $D = A^{\mathrm{T}}B \in \mathbb{R}^{3\times 3}$ и $E = AB^{\mathrm{T}} \in \mathbb{R}^{2\times 2}$.

Если необходимо перемножить массивы *поэлементно*, то следует использовать оператор «.*», как показано в следующем примере:

[1] Для матриц с комплексными элементами операция «’» указывает на *эрмитово сопряжение*: транспонирование матрицы с заменой элементов на комплексно-сопряженные. Операция транспонирования в общем случае задается оператором «.’».

```
D=A.*B
```

Получим:

```
D =
    10.     8.     24.
  - 7.      45.    0.
```

Конечно, эта операция выполнима только для массивов одинаковой размерности. Аналогично поэлементное деление массивов задается операцией «./», а возведение в степень каждого элемента массива – операцией «.^». Если первым операндом (сомножителем или делимым) является число, то надо следить за использованием символа «.». Так, результатом выполнения оператора:

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

будет

```
A  =
  - 2.      1.
    1.5   - 0.5
```

т. е. вычислена матрица $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}^{-1} = \begin{bmatrix} -2 & 1 \\ 1.5 & -0.5 \end{bmatrix}$.

Оператор

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

приведет к массиву, элементы которого обратны исходным:

```
 A  =
    1.           0.5
    0.3333333    0.25
```

Коэффициенты характеристического многочлена матрицы можно найти с помощью функции `poly`:

```
Q=poly(A,'q')
```

Получим:

```
Q =
                        2       3       4      5   6
    840 - 2038q + 1849q - 820q + 190q - 22q + q
```

Ниже перечислены еще несколько полезных функций:
`det(A)` – определитель матрицы A;
`inv(A)` – обратная к A матрица;
`trace(A)` – след (сумма диагональных элементов) матрицы A;
`spec(A)` – собственные числа и собственные векторы матрицы A.

1.4 Действия над многочленами

Создадим новый многочлен:

```
W=5-2*z+1.1*z^2;
```

и перемножим его с ранее созданным многочленом `P`:

```
S=P*W
```

Получим

```
S  =
                     2       3       4
    5 + 13z + 17.6z - 5.7z + 4.95z
```

Корни многочлена можно найти с помощью функции `roots`:

```
r=roots(s)
```

получим ответ:

```
r  =
  - 0.3333333 + 0.3333333i
  - 0.3333333 - 0.3333333i
    0.9090909 + 1.928473i
    0.9090909 - 1.928473i
```

Мы видим, что результат содержит мнимые числа.

Построим теперь график функции (многочлена) $S(z)$ для $z \in [0, 3]$ с шагом $\Delta z = 0.05$. Для этого выполним последовательность операторов

```
x=0:0.05:3;
y=horner(S,x);
xbasc();
plot(x,y)
```

Первой командой мы задали массив точек, в которых хотим вычислить значение многочлена $S(z)$. С помощью функции `horner` производится вычисление значений многочлена $S(z)$ в точках x и запись вычисленных значений в массив y. Далее с помощью функции `xbasc` очищается графическое окно. Затем с помощью функции `plot` получаем график функции $S(z)$.
Результат показан на рис. 1.1.

Деление многочленов осуществляется функцией `pdiv`. Разделим многочлен $W(z)$ на многочлен $P(z)$.

```
[R,Q]=pdiv(P,W)
```

Получим результат деления:

```
 Q  =
    4.0909091
 R  =
  - 19.454545 + 11.181818z
```

где Q – частное, а R – остаток от деления.

1.5 Пользовательские функции

1.5.1 Скрипт-файлы

Пользователь имеет возможность определить собственные программные модули, называемые *скрипт-файлами* (от англ. *script* – сценарий). Скрипт-файл не содержит заголовка и представляет собой последовательность команд. Эффект от его выполнения такой же, как если бы находящиеся в нем команды были набраны в командной строке. Создать скрипт-файл можно с помощью встроенного редактора Scipad или любого другого текстового редактора, например *Notepad* (Блокнот) в текстовом формате (*plain text*). Скрипт-файл запускается на исполнение функцией `exec`. Например, если мы создали и сохранили скрипт-файл под именем

script1.txt, то запустить его на исполнение следует строчкой exec('script.txt'). Если скрипт-файл не находится в рабочей директории, то в функции exec следует указать полный путь к нему.

1.5.2 Подпрограмма-функция

Определить собственную подпрограмму-функцию пользователь может оформив ее в виде файла с расширением sci, используя встроенный редактор Scipad или любой другой текстовый редактор. Имя созданного файла и определяет имя функции, которое будет использоваться при ее вызове. В отличие от скрипт-файла подпрограмма-функция должна содержать заголовок и оператор окончания. Заголовок идет в первой строке и начинается оператором function. После этого в заголовке указываются имена выходных переменных (если переменных несколько, то их имена перечисляются в квадратных скобках). Затем, после знака равенства, указывается имя функции и приводится заключенный в круглые скобки список имен входных (формальных) переменных, разделенных запятой. После заголовка идет само тело функции, за которым следует оператор окончания endfunction.

В качестве примера напишем подпрограмму, которая будет вычислять значения функции:

$$y(t) = \begin{cases} 1/\exp(t), & \text{если } t > 0, \\ \exp(t), & \text{иначе.} \end{cases} \qquad (1)$$

Для этого создадим редактором Scipad следующий файл:

```
function y=func(t)
y=zeros(t);
for j=1:length(t)
  if t(j)>0 then
    y(j)=1/%e^t(j);
  else
    y(j)=%e^t(j);
  end,
end
endfunction
```

Функцией `zeros` создается массив нулевых элементов такого же размера, как и входной массив t. Далее, внутри цикла `for` производится поэлементная проверка значений аргумента на выполнение условия (оператор `if`) и вычисление значений заданной функции. Сохраним файл в рабочей папке. Теперь вызовем функцию `func` из командного окна. Для этого выполним команду `exec('func.sci');` После этого ее можно использовать так же, как и стандартные функции языка Scilab.

1.6 Работа с графикой

1.6.1 Двумерные графики

В качестве первого примера вычислим значения функции (1) в точках отрезка $[-5, 5]$ с шагом 0.01 и построим ее график. Для этого выполним в командном окне следующую последовательность команд:

```
q=-3:0.001:3;
p=func(q);
xbasc();
xtitle('y(t)','t','y');
plot(q,p);xgrid;
```

Результат показан на рис. 1.2.

В качестве другого примера построим график значений многочлена и отобразим расположение его корней.

Зададим многочлен $W(z)$ и вычислим его корни:

```
W=1-5*%z^2+%z^3-2*%z^4+4*%z^5-%z^6;
s=roots(W)
```

Получим:

```
s  =
    0.4656907
  - 0.4070419
  - 0.4031434 + 0.9855944i
  - 0.4031434 - 0.9855944i
```

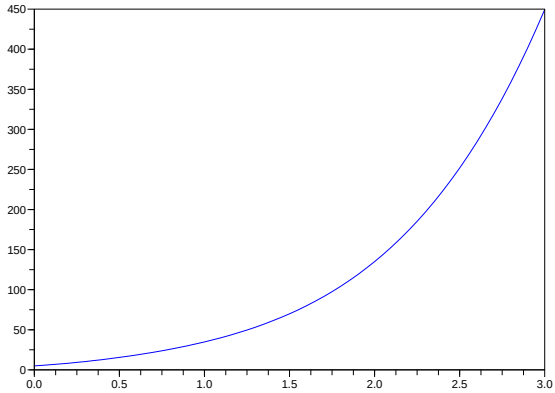

Рис. 1.1. График функции $S(z)$

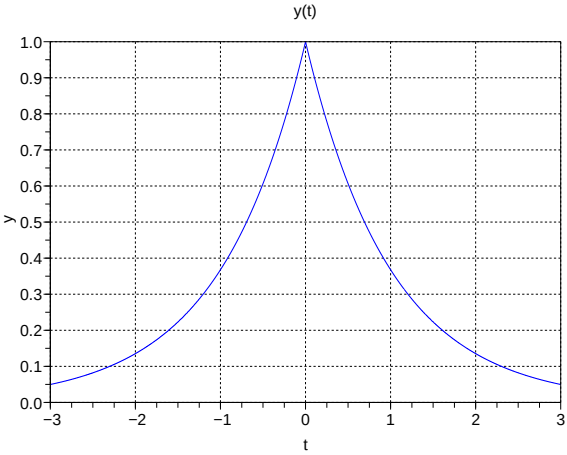

Рис. 1.2. График функции $y(t)$ вида (1)

```
    1.3825675
    3.3650705
```

Теперь найдем значения многочлена $W(z)$ на отрезке $z \in [0, 4]$ с шагом $\Delta z = 0.05$. Выполним следующие операторы:

```
x=0:0.05:4;
y=horner(W,x);
```

Отобразим полученные результаты графически:

```
xbasc();
subplot(121)
xtitle( 'Solution of W(z)', 'Re z', 'Im z');
xgrid();
plot(real(s),imag(s),'*');
subplot(122)
xtitle( 'Value W(z)', 'z', 'W(z)');
xgrid();
plot(x,y);
```

Результаты показаны на рис. 1.3. Оператор `subplot` используется, если нужно вывести несколько графиков в одном графическом окне. Так, запись `subplot(121)` означает, что график будет выведен в первой ячейке «таблицы» графиков, которая получена разделением графического окна на одну строку и два столбца (нумерация ячеек слева–направо, сверху–вниз). С помощью функции `xtitle` обозначены оси и заголовок графиков, а функция `xgrid` служит для нанесения сетки на график.

Многочлен $W(z)$ имеет комплексные корни. Чтобы выделить их вещественные и мнимые части, использованы встроенные функции `real(s)` и `imag(s)`.

Чтобы вывести несколько графиков в одном графическом окне, используется функция `xsetech`. Построим еще три графика. В качестве примера напишем в виде скрипт-файла следующие команды:

```
t=(0:0.05:5)';
xbasc();
xsetech([0,0,0.5,0.5]); plot2d(t, 2*sin(t));xgrid();
xsetech([0.5,0,0.5,0.5]); plot2d3(t, 1.5+cos(t));xgrid();
```

```
xsetech([0,0.5,1,0.5]); plot2d([t t], ...
[2*sin(t) 1.5+cos(t)],[12 10]);xgrid();
```

Теперь сохраним наш скрипт-файл под именем `graph` с расширением `sce`. Запустим его на выполнение командой:

```
exec('graph.sce');
```

Получим графики, показанные на рис. 1.4.

Функции `xsetech` в качестве аргумента передается вектор из четырех элементов, которые являются координатами графика относительно графического окна. Первые два элемента – координаты левого верхнего угла, а третий и четвертый – координаты правого нижнего угла. На втором графике с помощью функции `plot2d3` нарисована гистограмма, а на третьем показано, как можно построить несколько графиков на одной координатной сетке. Последним аргументом указаны цвета линий: число 12 соответствует синему цвету, а 10 – голубому.

1.6.2 Построение трехмерных изображений

Построим трехмерное графическое изображение тора. Поверхность тора в декартовых координатах (x, y, z) может быть задана параметрически через независимые аргументы α и β, принимающие значения от $-\pi$ до π следующим образом:

$$x = \sin\alpha(2 + \sin\beta),$$
$$y = \cos\alpha(2 + \sin\beta),$$
$$z = \cos\beta.$$

Составим программу:

```
[al,bet]=meshgrid(-%pi:%pi/20:%pi);
x=sin(al).*(2+sin(bet));
y=cos(al).*(2+sin(bet));
z=cos(bet);
```

Функция `meshgrid` создает двумерные массивы `al` и `bet` с заданным интервалом (в нашем примере – $[-\pi, \pi]$) и шагом изменения (здесь шаг равен $\pi/20$). Отобразим полученную поверхность графически последовательностью команд:

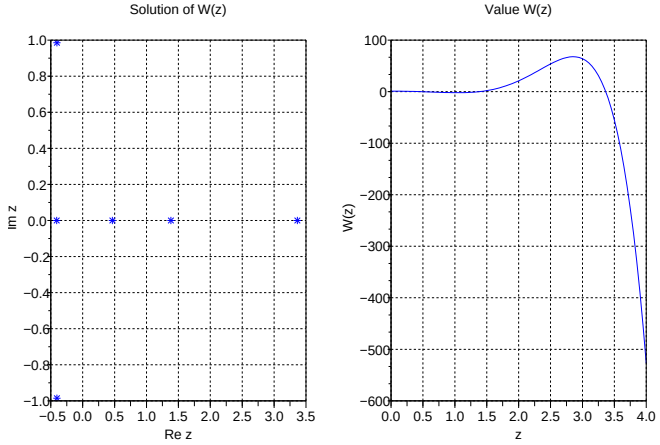

Рис. 1.3. Корни и значения многочлена $W(z)$

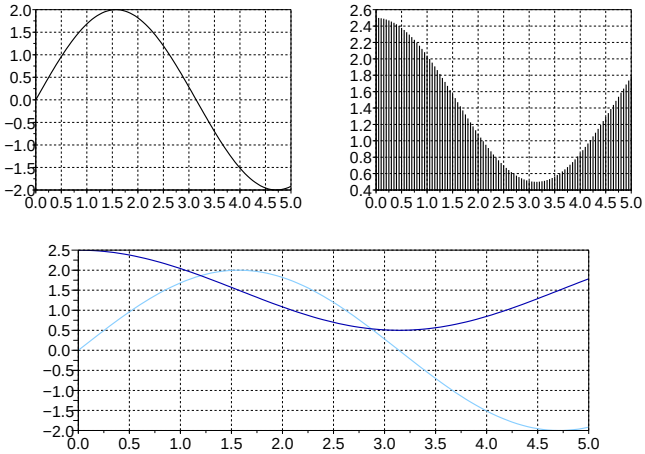

Рис. 1.4. Пример использования функции `xsetech`

```
xbasc();
xgrid();
mesh(x,y,z);
```

Функция `mesh` строит аксонометрическое изображение поверхности, показанное на рис. 1.5.

Функция, позволяющая строить раскрашенные трехмерные поверхности, – `surf`. Пример ее использования показан в следующей программе:

```
[x,y]=meshgrid(-10:0.5:10);
z=0.1*exp(-(0.05*x.^2+0.01*y.^2))-...
   0.7*exp(-(0.1*x.^2+0.3*y.^2))+exp(-(0.25*x.^2+0.9*y.^2));
surf(x,y,z,'facecol','interp');
```

Данной программой поверхность задана функцией $z = f(x, y)$ вида:

$$z = 0.1\exp(-(0.05x^2 + 0.01y^2)) - \\ - 0.7\exp(-(0.1x^2 + 0.3y^2)) + \exp(-(0.25x^2 + 0.9y^2)).$$

Результат показан на рис. 1.6. Свойством `facecol` задается схема затенения. Это свойство может быть установлено как `'flat'` (по умолчанию) – раскраска граней, или `'interp'` – выполняется билинейная цветовая интерполяция.

Теперь рассмотрим пример редактирования графика. Построим график поверхности $z = x^2 + y^2$. Выполним команды:

```
[x,y]=meshgrid(-3:0.25:3);
z=x.^2+y.^2;
xbasc();
mesh(x,y,z);
xset('window',1);
a=gca();
a.grid=[1 2 -1];
a.labels_font_size=3;
a.x_label.text='x';
a.x_label.font_size = 4;
a.y_label.text='y';
a.y_label.font_size = 4;
a.z_label.text='z';
```

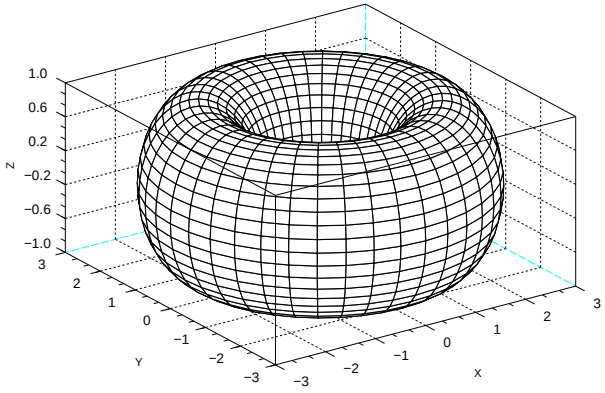

Рис. 1.5. Поверхность тора

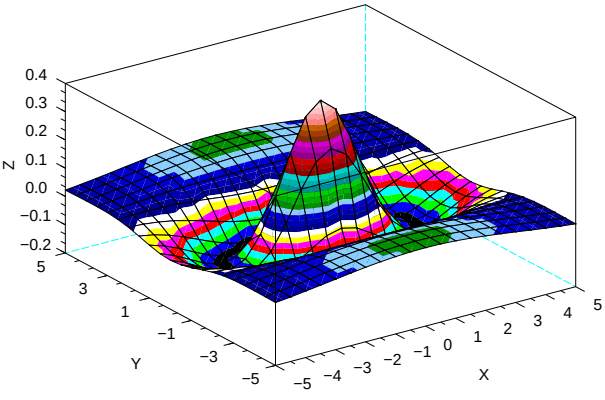

Рис. 1.6. Использование функции `surf`

```
a.z_label.font_size = 4;
a.title.text="z=x^2+y^2";
a.title.font_size = 4;
mesh(x,y,z);
a.isoview="on";
```

В результате (рис. 1.7) построено два графика одной поверхности, первый – с настройками по умолчанию, второй – отредактированный. Команда `xset('window',1)` использована для того, чтобы вывод второго графика был произведен в новое графическое окно. Командой `gca` был получен описатель (*handle*) осей графика, через который можно менять их свойства. Командой `a.grid=[1 3 -1]` устанавливается координатная сетка по осям , y и z соответственно: по оси сетка черного цвета, по оси y – зеленого, а по оси z вообще отсутствует. Свойство `labels_font_size` устанавливает размер шрифта шкалы. Далее были установлены названия и размеры шрифта осей и графика.

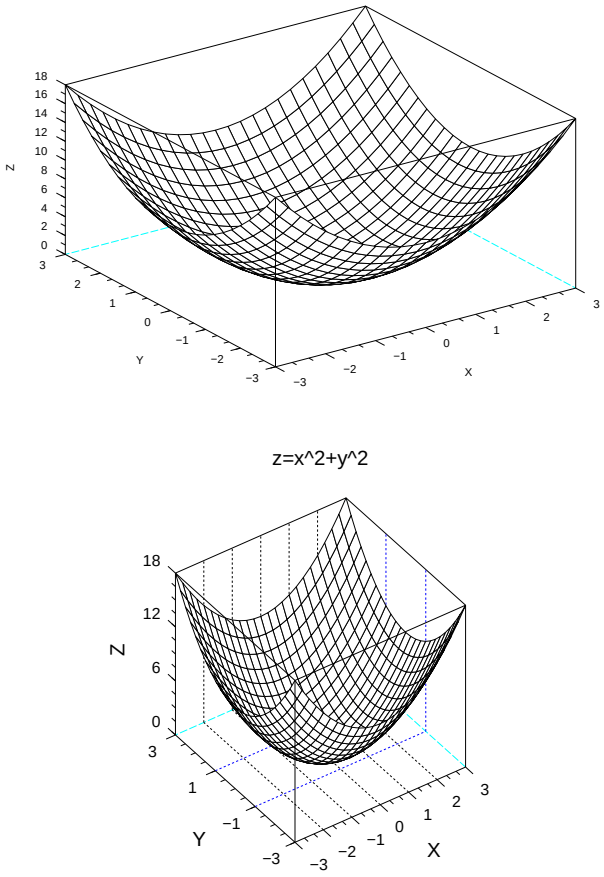

Рис. 1.7. Пример редактирования изображения

Глава 2

Примеры исследования динамических систем

2.1 Расчет частотных характеристик

Рассмотрим электрическую цепь, изображенную на рис. 2.1 [1, 6]. Эта RLC-цепь представляет собой колебательный контур. Выходным сигналом $y(t)$ считаем напряжение на зажимах индуктивного элемента $u_L(t)$, а входом – падение напряжения на всей цепи $u(t)$. Как известно из электротехники, выполнены соотношения: $u_L(t) = L\dfrac{\mathrm{d}i(t)}{\mathrm{dt}}$, $i(t) = C\dfrac{\mathrm{d}u_C(t)}{\mathrm{dt}}$ и $u(t) = u_L(t) + i(t) + R + u_C(t)$. Отсюда получаем дифференциальное уравнение:

$$LC\frac{\mathrm{d}^2y(t)}{\mathrm{dt}^2} + RC\frac{\mathrm{d}y(t)}{\mathrm{dt}} + y(t) = LC\frac{\mathrm{d}^2u(t)}{\mathrm{dt}^2},$$

а также *передаточную функцию* цепи:

$$W(s) = \frac{Ks^2}{T^2s^2 + 2\xi Ts + 1},$$

где $T = \sqrt{LC}$, $K = LC = T^2$, $\xi = \dfrac{R}{2}\sqrt{\dfrac{C}{L}}$. Аргументом передаточной функции является комплексная переменная $s \in \mathbb{C}$.

Исследуем эту систему численно. Для этого примем следующие значения параметров: $R = 800$ Ом, $L = 4$ Гн, $C = 10^{-5}$ Ф. Выполним следующие команды:

```
R=800 ;
L=4;
C=1e-5;
T=sqrt(L*C);
K=L*C;
xi=R/2*sqrt(C/L);
omega=0:0.1:600;
s=%i*omega;
W=(K*s.^2)./(T^2.*s.^2+2*xi*T.*s+1);
```

Последним оператором были получены численные значения *частотной характеристики* $W(i\omega)$ рассматриваемой цепи. С этой целью оператором `omega=0:0.1:600` введен массив значений *частоты* $\omega \in [0, 600]$ с шагом $\Delta\omega = 0.1$ (все значения приведены в системе СИ), затем оператором `s=%i*omega` для всех ω сформирован массив аргументов $s = i\omega$ (где i – мнимая единица) и оператором `W=(K*s.^2)./(T^2.*s.^2+2*xi*T.*s+1)` найдены значения *частотной передаточной функции* $W(i\omega)$. Заметим, что в последних двух операторах программы выполняются действия над *массивами в целом*, что позволяет избежать использования оператора цикла. Чтобы деление массивов выполнялось поэлементно (а не трактовалось как недопустимая в данном случае операция обращения матриц), в последнем операторе использован знак «`./`», а не «`/`».

Построим частотные характеристики системы. *Амплитудно-частотная характеристика* (АЧХ) определяется как модуль частотной передаточной функции системы:

$$A(\omega) = |W(i\omega)| = \frac{K\omega^2}{\sqrt{(1 - T^2\omega^2)^2 + 4\xi^2 T^2 \omega^2}}$$

Вычислим значения АЧХ:

```
A=abs(W);
clf();
plot(omega,A); xgrid();
xtitle('A(omega) = | W(i*omega) |','omega, 1/c','A(omega)');
```

и построим ее график (рис. 2.2).

Теперь построим график *фазочастотной характеристики* $\varphi(\omega)$. Известно, что $W(i\omega) = A(\omega)e^{i\varphi(\omega)}$, где $A(\omega) = |W(i\omega)|$,

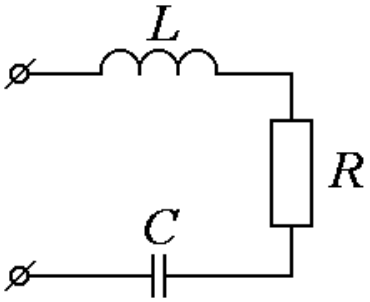

Рис. 2.1. LRC-цепь

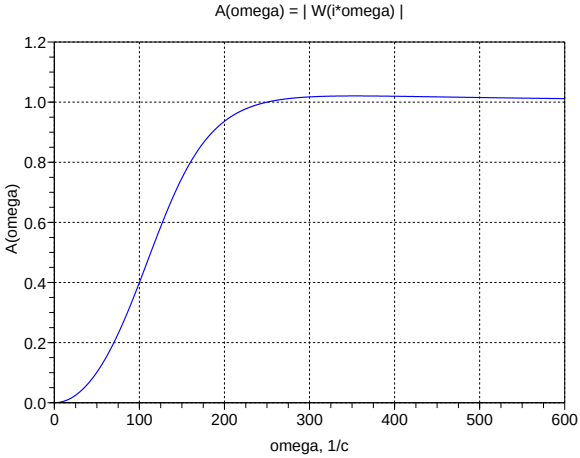

Рис. 2.2. Амплитудно-частотная характеристика колебательного звена

$\varphi(\omega) = \arg W(i\omega)$. Таким образом, фазочастотная характеристика для RLC-цепи при $\omega \geq 0$ будет такой:

$$y(t) = \begin{cases} \pi - \operatorname{arctg} \dfrac{2\xi T\omega}{1 - T\omega^2} & \text{при } T\omega^2 > 0, \\ \dfrac{\pi}{2} \quad \text{при } T\omega^2 = 1, \\ \operatorname{arctg} \dfrac{2\xi T\omega}{-1 + T\omega^2} & \text{при } T\omega^2 < 0. \end{cases}$$

Реализуем эту функцию с помощью следующего набора операторов:

```
for i=1:length(omega)
if (T*omega(i)^2) < 1
    then phi(i)=%pi-atan(2*xi*T*omega(i)/(1-K*omega(i)^2));
elseif (T*omega(i)^2)>1
    then phi(i)=atan(2*xi*T*omega(i)/(-1+K*omega(i)^2));
elseif (T*omega(i)^2)==1
    phi(i)=%pi/2;
end,
end
```

Замечание. Этот пример служит для демонстрации *условного оператора* `if-elseif-end`. Вычисление частотной характеристики для данного примера можно было бы выполнить без этого оператора с использованием функции `atan(y,x)` с двумя аргументами (аналогичной функции `atan2` в ряде языков программирования, таких как фортран и Matlab), возвращающей аргумент комплексного числа $x + iy$ и лежащий в пределах $(-\pi, \pi]$ [1,3–5].

Теперь выведем полученный результат на график:

```
clf();
plot(omega,phi);xgrid();
xtitle('phi (omega) = arg W(i*omega)',...
'omega, 1/c','phi (omega)');
```

приведенный на рис. 2.3.

Построим графики *вещественной и мнимой частотных характеристик* $U(\omega) = \operatorname{Re} W(i\omega)$, $V(\omega) = \operatorname{Im} W(i\omega)$ и *кривую Найквиста* (*АФХ*). Для этого выполним следующие команды:

```
clf();
U=real(W);
V=imag(W);
subplot(121);
plot(omega,U,'r',omega,V,'b'); xgrid();
xtitle('','omega, 1/c','U(omega), V(omega)');
subplot(122);
plot(U,V); xgrid();
xtitle('','ReW','ImW');
```

Получим графики, показанные на рис. 2.4.

Рассмотрим построение *диаграммы Боде* (*логарифмическая амплитудно-частотная характеристика* — ЛАЧХ). Как известно, она определяется как $L(\omega) = 20 \lg A(\omega)$, измеряется в децибелах и строится в функции от $lg(\omega)$. Выполним программу:

```
clf();
omega=0.1:1/6:600;
s=%i*omega;
W=(K*s.^2)./(T^2.*s.^2+2*xi*T.*s+1);
A=abs(W);
La=20*log10(A);
plot2d1('oln',omega,La); xgrid();
xtitle('L=20lgA(omega)','omega, 1/c','L, dB');
```

Полученная диаграмма Боде (ЛАЧХ) для RLC-цепи представлена на рис. 2.5. График в логарифмическом масштабе построен с помощью функции `plot2d1`. Первый аргумент этой фунции – строка из трех символов, которая определяет вид графика. Первый символ *o* означает, что в случае построения нескольких кривых на одном графике аргумент x для них будет один и тот же. Следующие два символа определяют способ масшабирования осей x и y (соответственно): l означает логарифмический масштаб, n – линейный.

Частотные характеристики системы можно также получить и с помощью функции `syslin`. Рассмотрим ее применение на примере `дифференцирующего звена с замедлением`, передаточная функция которого имеет вид:

$$W(s) = \frac{Ks}{Ts+1},$$

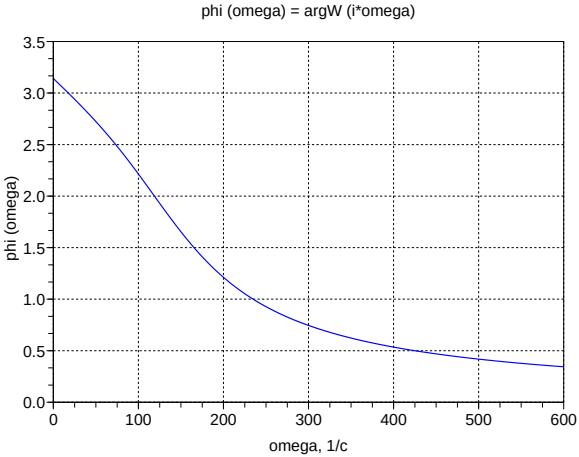

Рис. 2.3. Фазочастотная характеристика колебательного контура

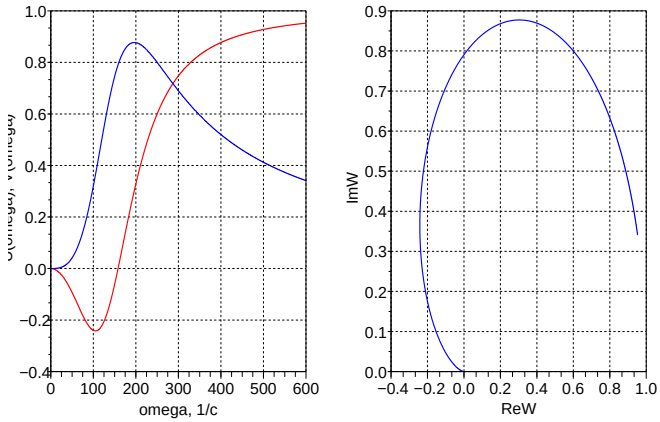

Рис. 2.4. Вещественная и мнимая частотные характеристики и кривая Найквиста.

где K, T – параметры, $s \in \mathbb{C}$.

Построим диаграмму Боде и годограф Найквиста такой системы при $T = 0.3$ и $K = 5 \cdot 10^{-3}$. Для этого выполним операторы:

```
K=0.3;
T=5e-3;
s=poly(0,'s');
w=syslin('c',K*s/(T*s+1));
clf();
subplot(121);
bode(w,0.01,1000);
subplot(122);
nyquist(w,0.01,1000);
```

Первый параметр функции `syslin` определяет тип системы: непрерывная («`c`») или дискретная («`d`»). Вторым параметром задаем передаточную функцию данного звена. Для этого вначале введена переменная `s` в качестве аргумента многочлена (см. п. 1.2). С помощью функции `bode` построена диаграмма Боде (ЛАХ и ЛФЧХ), а с помощью функции `nyquist` – кривая Найквиста (АФХ). Эти функции имеют одинаковые наборы входных параметров: первым параметром передается имя системы, созданной при помощи оператора `syslin`, а вторым и третим аргументами – минимальная и максимальная частоты соответственно. Результат показан на рис. 2.6.

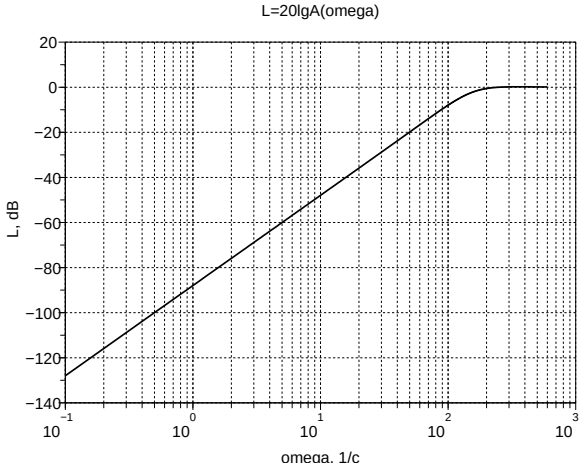

Рис. 2.5. Диаграмма Боде для RLC-цепи

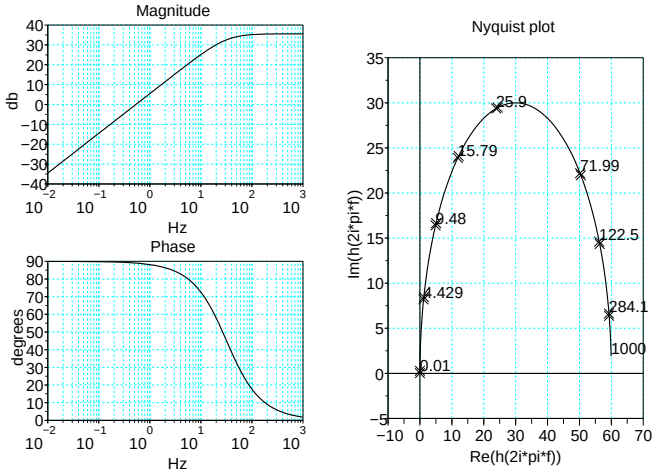

Рис. 2.6. Использование функций `bode` и `nyquist`

2.2 Расчет переходных процессов

Задачи моделирования часто сводятся к интегрированию систем обыкновенных дифференциальных уравнений (ОДУ) первого порядка вида

$$\dot{y}(t) = f(t, y), \quad y(t_0) = y_0,$$

где $y(t)$ – n-мерный вектор, $y = [y_1, ..., y_n]^{\mathrm{T}}$; t – независимая переменная, $f(t, y)$ – n-мерная вектор-функция, $f(t, y) = [f_1(t, y), \ldots, f_n(t, y)]^{\mathrm{T}}$; t_0 – начальный момент времени; $y_0 \in \mathbb{R}^n$ – вектор начальных условий.

Для решения ОДУ используется функция Scilab `ode`.

В качестве первого примера рассмотрим RLC-цепочку, входом будем считать падение напряжения на всей цепи, а выходом – заряд конденсатора $q(t)$. Из второго закона Кирхгофа следует:

$$L\frac{\mathrm{d}i}{\mathrm{dt}} + Ri + \frac{1}{C}q = u, \quad q = \int_0^t i(s)ds,$$

откуда получим систему линейных дифференциальных уравнений первого порядка:

$$\frac{\mathrm{d}q}{\mathrm{dt}} = i, \quad \frac{\mathrm{d}i}{\mathrm{dt}} = -\frac{R}{L}i - \frac{1}{LC}q + \frac{1}{L}u.$$

Проинтегрируем ее при помощи функции `ode`. Для интегрирования следует написать головную программу, содержащую обращение к процедуре `ode` и подпрограмму-функцию, в которой вычисляются правые части системы дифференциальных уравнений. В головной программе должны быть заданы начальные условия для всех переменных, начальное и конечное значения аргумента интегрирования. Текст программы приведен ниже:

```
R=2;
L=0.35;
C=0.25;
Q=[0 1;-1/(L*C) -R/L];
P=[0;1/L];
deff('dy=F(t,y)','dy=Q*y+P*u(t)');
```

```
deff('[ut]=u(t)','ut=sin(4*t)');
y0=[0;0];t0=0;
T=0:0.05:5;
Y=ode(y0,t0,T,F);
plot2d1("onn",T',Y',[-1 1]); xgrid();
xtitle('','t','i, q');
```

Вместо того, чтобы описывать подпрограмму-функцию в отдельном файле, можно описать ее непосредственно в головной программе с помощью функции `deff`, как это сделано в приведенном примере. Первым параметром этой функции является строка вида `[s1,s2,...]=newfunction(e1,e2,....)`, где `s1,s2,...` – выходные переменные, `e1,e2,...` – входные переменные, а `=newfunction` – имя описываемой функции. Второй аргумент – строка, определяющая последовательность инструкций создаваемой функции. Таким образом, команда:

```
deff('dy=F(t,y)','dy=Q*y+P*u(t)');
```

определяет функцию, аналогичную функции, описываемой в отдельном файле:

```
function dy=F(t,y)
dy=Q*y+P*u(t);
endfunction
```

В таком случае переменные `Q` и `P` необходимо определить как глобальные или определять в самой подпрограмме.

В рассмотренной программе были описаны две функции: первая вычисляет правые части системы дифференциальных уравнений, а вторая задает синусоидальное входное воздействие. Входными параметрами функции `ode` являются `y0` – вектор начальных состояний, `t0` – начальный момент времени, `T` – моменты времени, для которых производится вычисления, `F` – подпрограмма-функция, вычисляющая правые части системы дифференциальных уравнений. Выходным аргументом в данном примере будет вектор $Y = (i, q)$. Полученные результаты отображены на рис. 2.7.

Часто в подпрограмму нужно передавать дополнительные аргументы. Так, для функции `dy=f(t,y,u1,...,un)` требуются дополнительные параметры `u1,...,un`. Эти аргументы можно передать при вызове функции `ode` как список вида `list(f,u1,...un)`.

Рассмотрим нелинейную систему:

$$\dot{y}_1(t) = y_2(t) + \big(a - \|y(t)\|\big)y_1(t), \tag{1}$$

$$\dot{y}_2(t) = -y_1(t) + \big(a - \|y(t)\|\big)y_2(t). \tag{2}$$

Напишем следующую подпрограмму вычисления правых частей системы: (2)

```
function dy=fun(t,y,a)
dy(1)=y(2)+(a-norm(y))*y(1);
dy(2)=-y(1)+(a-norm(y))*y(2);
endfunction
```

Ниже приведен текст головной программы:

```
exec('fun.sci');
t0=0;y0=[-3;3];
T=0:0.1:10;
a=2;
y=ode(y0,t0,T,fun,list(fun,a));
plot(y(1,:),y(2,:)); xgrid();
```

Фазовая траектория системы (2) на плоскости (y_1, y_2) показана на рис. 2.8.

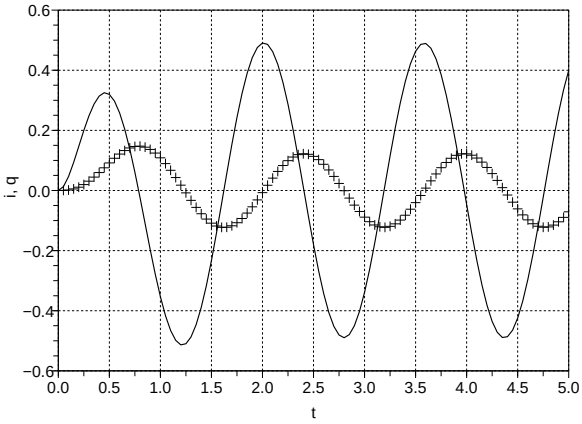

Рис. 2.7. Моделирование RLC-цепочки

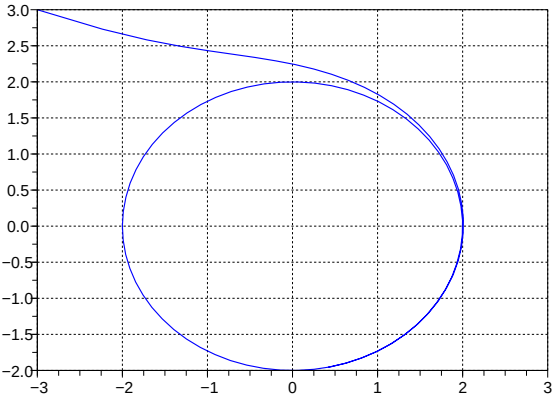

Рис. 2.8. Фазовая траектория системы (2)

Глава 3

Система визуального моделирования XCOS

3.1 Модель двигателя постоянного тока

Инструмент визуального моделирования Xcos служит для создания структурных схем математических моделей, и выполнения соответствующих действий над ними. Для вызова необходимо набрать xcos в командной строке или кликнуть иконку на панели инструментов в главном окне. Откроется два окна: палитры блоков и рабочее пространство. В качестве примера построим модель двигателя постоянного тока, представив его как систему второго порядка. Каждый блок находится в своей палитре. Для начала найдем в палитре *Системы с непрерывным временем* интегратор, блок называется *Integral*. Далее понадобятся статическое звено и сумматор,блоки *Gainblk* и *Bigsom* в палитре *Математические операции*. Каждый блок может быть перемещен мышью в поле диаграммы. Параметры любого блока вызываются сочетанием клавиш «Ctrl+B». Далее необходимо собрать диаграмму представленную на рис. 3.1.

Численные значения взяты из реальных параметров двигателя постоянного тока без учета редуктора. Теперь надо добавить блоки *Clock* и *Pulse* из палитры *Источники сигналов и воздействий* для моделирования управляющего ШИМ-сигнала и блок *Cscope* из палитры *Регистрирующие устройства*. Получим диаграмму, представленную на рис. 3.2.

Блок *Cscope* используется для построения графиков нескольких

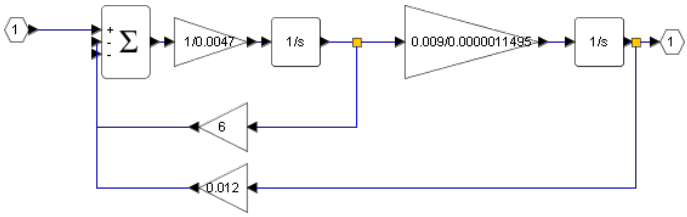

Рис. 3.1. Модель двигателя постоянного тока.

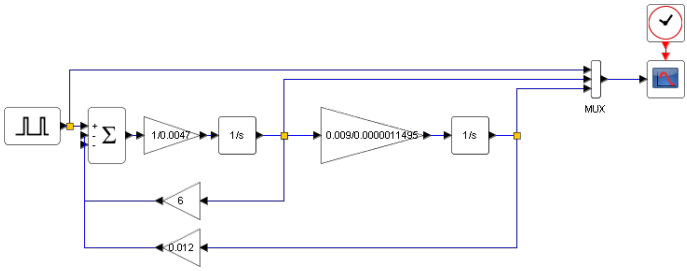

Рис. 3.2. Управление двигателем постоянного тока посредством ШИМ.

сигналов в одном графическом окне. *Clock* генерирует временную переменную, которая имеет два параметра: начальное значение и шаг дискретизации, в данном случае 0.000001. Блоку *Pulse* зададим параметры, согласно рис. 3.3.

Параметры соответствуют следующим значениям: сдвиг фазы, ширина импульса, период и амплитуда. Параметры моделирования задаются в диалоговом окне Моделирование – Установка. Установим конечное время интегрирования равное 0.001, а в параметрах блока *Cscope* зададим значения, как указано на рис. 3.4. Вычисления выполняются кнопкой *Запустить* на панели инструментов.

Если мы хотим представить двигатель одним блоком, то можно воспользоваться блоком *Super* в палитре *Порты и подсистемы*. Вместо параметров блока появится новое поле, в которое можно добавить дополнительные элементы входа и выхода из той же палитры. На поле блока *Super* собираем модель двигателя и подключаем управляющие входы и выходы системы. На рис. 3.5 представлена схема сравнивающая ВСВ и ВВ модели двигателя.

Теперь рассмотрим пример использования блока *Scifunc*, находящегося в разделе *Пользовательские функции*. Этот блок позволяет преобразовывать входной сигнал в соответствии с заданной пользователем функцией. Это может быть и стандартная функция Scilab, и функция, описаная пользователем. Построим модель следящей системы, в которой датчик рассогласования имеет зону нечуствительности, а усилитель – ограниченную зону линейности, т. е. насыщение. Эти две нелинейности можно описать одной нелинейной зависимостью (рис. 3.7), где $f(\sigma)$ – нелинейная функция, σ_1 – ширина зоны нечуствительности, $\bar{u}$ – уровень насыщения. При $\bar{u} \to \infty$ и $\sigma_1 \to 0$ приходим к линейной модели. Передаточная функция линейной части:

$$W_{lin}(s) = \frac{k_l(T_2 s + 1)}{s(T_1 s + 1)(T_3 s + 1)(T_4 s + 1)}$$

Xcos-модель системы представлена на рис. 3.8. Опишем нелинейную часть системы в виде функции:

```
function y1=funci(u1)
sig2=sig1-u_/k_;
if abs(u1) <= sig1 then
```

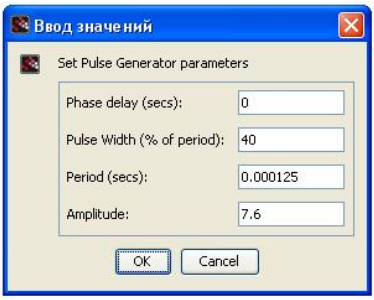

Рис. 3.3. Окно параметров блока «PULSE».

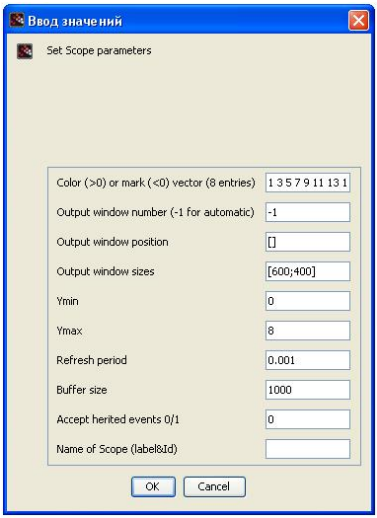

Рис. 3.4. Окно параметров блока CSCOPE.

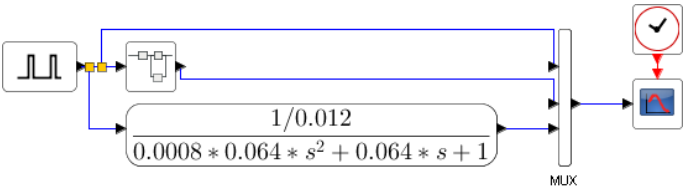

Рис. 3.5. Пример использования блока «SUPER».

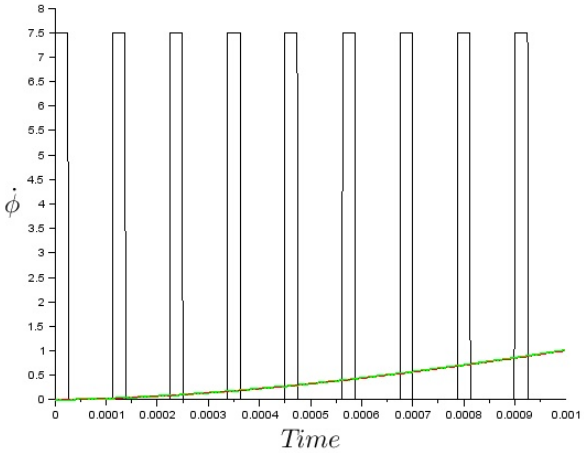

Рис. 3.6. Графики управляющего воздействия и скорости вращения двигателя.

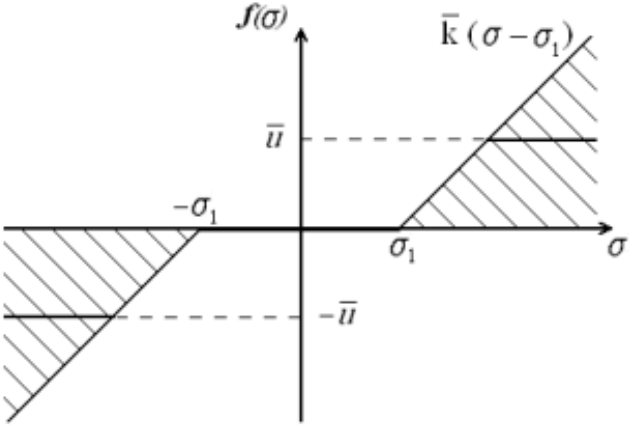

Рис. 3.7. Нелинейная часть следящей системы

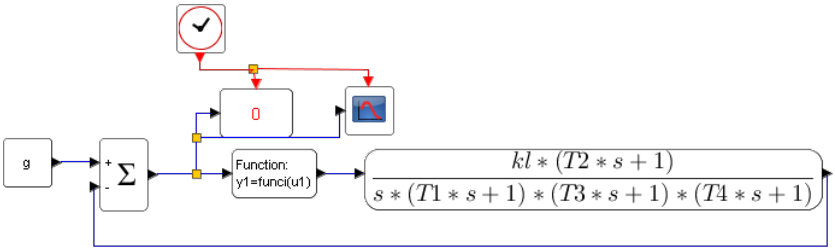

Рис. 3.8. Использование блока *Scifunc* для построения модели нелинейной системы

```
y1=0;
elseif abs(u1) < sig2 then
y1=k_*(u1-sign(u1)*sig1);
else
y1=u_*sign(u1);
end
endfunction
```

Сохраним ее в рабочей папке как `nlin.sci`. Выполним эту функцию в главном окне Scilab командой `exec('nlin.sci')`. Теперь следует настроить блок *Scifunc*. При выборе его свойств появляется диалоговое окно, в котором можно установить такие параметры блока, как количество входных и выходных портов, начальные состояния и др. Оставим эти настройки по умолчанию и нажмем кнопку `Ok`. В следующем диалоговом окне предлагается определить функцию, которая будет вычислять выходной сигнал. В этом окне введем строчку `y1=nlin(u1)`. Это означает, что выходной сигнал порта `y1` будет вычислен по функции `nlin` с сигналом входного порта `u1` в качестве аргумента. Следует заметить, что описать необходимую функцию можно прямо в этом диалоговом окне, не описывая ее в отдельном файле. Далее определим значения параметров системы (Edit – Context):

```
g=50;
T1=1; T2=0.25; T3=0.15; T4=0.03;
kl=4;
k_=25; sig1=5; u_=35;
```

В блоке `Continuous transfer function` зададим передаточную функцию линейной части системы, а значение переменной `g` укажем в блоке `Const`. Запуская процесс моделирования, получим график переходного процесса в следящей системе.

3.2 Исследование автоколебаний в нелинейной системе

Одной из особенностей нелинейных систем является возможность возникновения автоколебаний. Это зависит в общем случае от пара-

метров нелинейностей, структуры линейной части и значений отдельных параметров динамических звеньев системы, а также от начальных условий. Рассмотрим нелинейную САУ (рис. 3.9) и построим ее модель в среде Xcos (рис. 3.10).

Выполним моделирование системы при параметрах $b = 20$, $c = 20$, $k = 5$, $T_1 = 2$, $T_2 = 1$ и $g_0 = 10$. Видно возникновение автоколебаний в системе (рис. 3.11).

3.3 Исследование системы с переменной структурой

Системы с переменной структурой (СПС) – специальный класс нелинейных систем, в котором происходит переключение регуляторов или их параметров по сигналам блока изменения структуры в зависимости от значений переменных состояния объекта [6, 8, 9]. Обычно СПС состоит из двух линейных регуляторов и блока изменения структуры, работающего по релейному закону. Структурная схема такой системы приведена на рис. 3.12. Здесь Р1 и Р2 – регуляторы, $g(t)$ – задающее воздействие, $y(t)$ – выход системы, $u(t)$ – управляющее воздействие, БИС – блок изменения структуры, ОУ – объект управления, ИУ – исполнительное устройство. Рассмотрим СПС, объект управления которой описывается уравнениями:

$$\dot{x}_1 = x_2, \quad \dot{x}_2 = -ax_1 + bu.$$

Построим модель системы (рис. 3.13). Модель объекта управления набрана из двух интеграторов и блоков *Sum* и *Gain*. Условие переключения $x_1 \cdot (0.8x_2 + x_1) \geq 0$ фиксируется блоком произведения *Product*.

Закон управления реализован блоками *Switch*, *Filter* и *Inv*. Блок *Filter* представляет собой апериодическое звено первого порядка с передаточной функцией $W(s) = \dfrac{1}{0.02s + 1}$, а блок *Inv* – усилитель с коэффициентом передачи -1. Блок *Switch* выполняет операцию переключения значения выхода с 0 на $-x$ в зависимости от знака сигнала на выходе блока *Product*. Снятие фазового портрета осуществляется блоком *XY Scope*. Начальное состояние системы задано на интеграторах `Int1` и `Int2` $x_1(0) = 0, x_2(0) = 10$ (параметр Initial

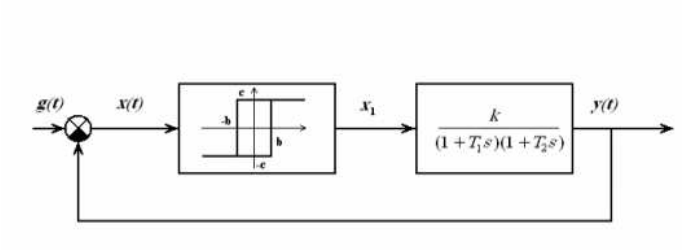

Рис. 3.9. Структурная схема нелинейной системы

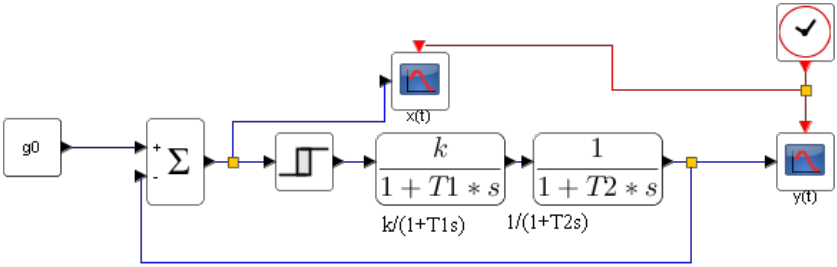

Рис. 3.10. Структурная схема автоколебательной нелинейной системы в среде Xcos

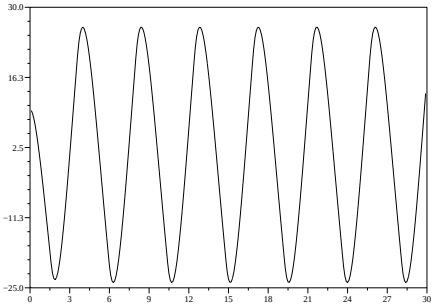

Рис. 3.11. Автоколебательный процесс в нелинейной системе

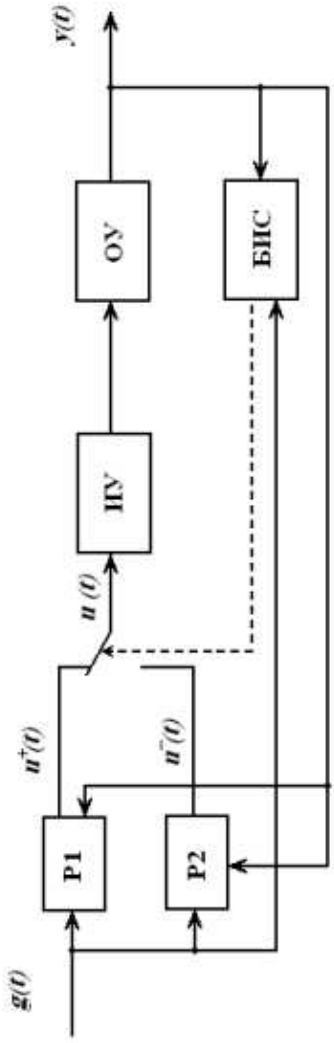

Рис. 3.12. Структурная схема системы с переменной структурой

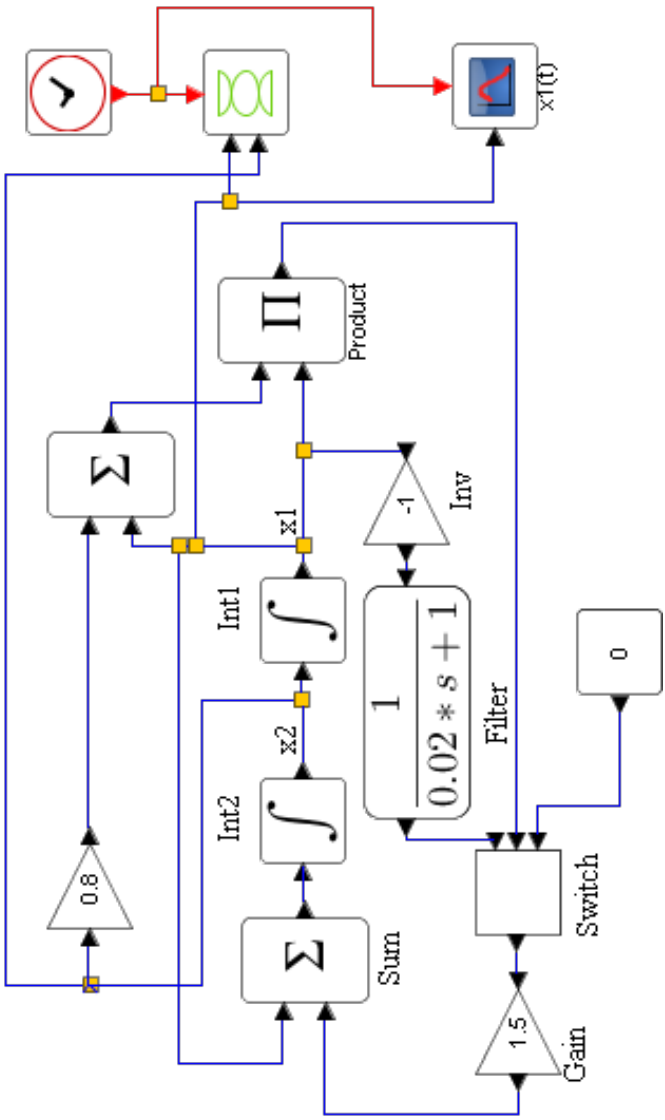

Рис. 3.13. Модель системы с переменной структурой

Condition). Время окончания моделирования установлено равным 30 с. Результатом моделирования будет фазовый портрет системы и график сигнала $x_1(t)$ (рис. 3.14).

Глава 4

Тулбокс SYSTEMS AND CONTROL

4.1 Описание линейных динамических систем

Для решения различных задач управления важную роль играют линейные динамические модели систем, позволяющие использовать широко развитый математический аппрарат. Линейные системы – базовый объект Scilab Control Toolbox [5]. Существует несколько способов определения линейных моделей систем и действий над ними.

Различают непрерывные и дискретные линейные системы [1, 6, 8]. Уравнения состояния для непрерывных систем имеют вид:

$$\begin{aligned} &\dot{x}(t) = Ax(t) + Bu(t), \quad x(0) = x_0; \\ &y(t) = Cx(t) + Du(t), \quad t \in \mathbb{R}. \end{aligned} \tag{1}$$

Соответственно, для дискретной линейной системы:

$$x(t+1) = Ax(t) + Bu(t), \quad x(0) = x_0 \tag{2}$$

$$y(t) = Cx(t) + Du(t), \quad t \in \mathbb{Z}. \tag{3}$$

Непрерывные и дискретные линейные системы представлены в Scilab как абстрактные объекты и могут быть определены с помощью функции `syslin`. Выполним следующие операторы:

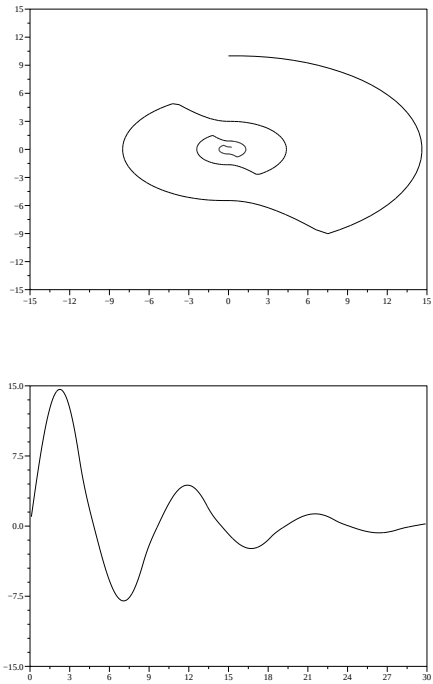

Рис. 3.14. Фазовый портрет и процесс $x_1(t)$
в системе с переменной структурой

```
A=diag([2, 3, 4]); B=[1 0; 0 1; 0 0];
C=[1 -1 0]; D=0*C*B; x0=[0; 0; 0];
Sl=syslin('c',A,B,C,D,x0)
Получим
Sl =
l(1) (state-space system:)
lss
Sl(2) = A matrix =
! 2. 0. 0. !
! 0. 3. 0. !
! 0. 0. 4. !
Sl(3) = B matrix =
! 1. 0. !
! 0. 1. !
! 0. 0. !
Sl(4) = C matrix =
! 1. -1. 0. !
Sl(5) = D matrix =
! 0. 0. !
Sl(6) = X0 (initial state) =
! 0. !
! 0. !
! 0. !
Sl(7) = Time domain =
c
```

Этими командами были заданы матрицы A, B, C и D параметров системы, вектор начальных состояний x_0 и определена непрерывная линейная система `Sl`. Первый параметр функции `syslin` определяет дикретную или непрерывную систему: символ `"c"` – непрерывную, а символ `"d"` – дискретную, а также может определять период дискретности. Например команда `syslin(0.2,A,B,C)` определяет дискретную систему с периодом дискретности 0.2 с, т. е. с частотой квантования $f_s = \dfrac{1}{0.2} = 5$ Гц. Структура `Sl` с точки зрения языка – типизированный список. Таким образом, аналогичный результат можно было получить командой `sl=tlist(['lss','A', 'B', 'C', 'D','X0', 'dt'], A,B, C,D, x0,"c")`. Преобразовать систему из непрерывной в дискретную можно следующим обра-

зом: `Sl("dt")="d"`. Матрицу передаточных коэффициентов $H(s) = C(sI - A)^{-1}B + D$ можно получить с помощью функции `ss2tf`:

```
Slt=ss2tf(Sl)
```

Получим:

```
Slt  =
      1       - 1
    -----     -----
  - 2 + s   - 3 + s
```

Соединение систем производится с помощью операторов `+`, `-`, `/`, а также `[` и `]`.

4.2 Типовые соединения линейных систем

Рассмотрим основные соединения (рис. 4.1) на примере двух линейных динамических систем:

```
K=1;
T=3;
S1=syslin('c', K/%s)
S2=syslin('c', K/T^2%s^2+1)
```

Полученые системы:

```
S1  =
    1
    -
    s
S2  =
      1
    -----
          2
    1 + 9s
```

`S1` соответствует интегрирующему звену, а `S2` – консервативному звену. Последовательное соединение (рис. 4.1, *а*):

```
S=S2*S1
```

Получим:

```
S  =
       1
    -----
          3
    s + 9s
```

Параллельное соединение (рис. 4.1, *б*):

```
S=S2+S1
```

Получим:

```
S  =
                2
    1 + s + 9s
    ----------
            3
      s + 9s
```

Соединение с обратной связью (рис. 4.1, *в*):

```
S=S1/.S2
```

Получим:

```
S  =
              2
      1 + 9s
    ----------
               3
    1 + s + 9s
```

Система с одним входом и двумя выходами (рис. 4.1, *г*):

```
S=[S1; S2]
```

Получим:

```
S  =
    1
    -
    s

      1
    -----
          2
    1 + 9s
```

Система с двумя входами и одним выходом (рис. 4.1, *д*):

```
S=[S1,S2]
```

Получим

```
S  =

    1       1
    -     -----
                2
    s     1 + 9s
```

4.3 Синтез наблюдателя состояния

Рассмотрим линейную систему, описанную матрицами (A, B, C, D) с внутренней переменной x, входом u и выходом y. При входном сигнале $z = Hx$ линейный наблюдатель – линейная система $(u, y) \to \hat{z}$, такая, что ошибка $z(t) - \hat{z(t)}$ стремится к нулю при $t \to \infty$. Синтезировать наблюдателя для наблюдаемой системы можно с помощью функции `observer`. Рассмотрим использование этой функции для модели следующей нелинейной системы:

$$\begin{aligned} \dot{x}_1 &= p_r x_1 (1 - x_1/p_k) - u p_a x_1 x_2, \\ \dot{x}_2 &= p_s x_2 (1 - x_2/p_l) - u p_b x_1 x_2. \end{aligned} \tag{4}$$

Зададим модель с помощью функции:

```
function dx=competition(t,x,u)
```

```
dx=[p_r*x(1)*(1-x(1)/p_k)-u*p_a*x(1)*x(2) ; ...
    p_s*x(2)*(1-x(2)/p_l)-u*p_b*x(1)*x(2)]
endfunction
```

Параметры системы приняты следующими:

```
p_r=0.01; p_a=5d-5;
p_s=5d-3; p_b=1d-4;
p_k=1d3; p_l=500;
u_e=1;
```

Точки равновесия можно вычислить с помощью функции `fsolve`, позволяющей вычислить решение системы нелинейных уравнений:

```
deff('[y]=f(x)','y=competition(0,x,u_e)')
x_i=[20,200];
x_e=fsolve(x_i,f);
```

Следует линеаризовать исходную систему. Линеаризованная модель может быть получена с помощью функции `tangent`. Вызывающая последовательность этой функции:

```
[f,g,newm]=tangent(ff,xe,[ue]),
```

где `ff` – строка-название функции Scilab, в которой описана линеаризуемая система, `xe` – вектор-столбец, в котором передаются точки равновесия системы для значения параметра `ue`. `f` и `g` – матрицы для линеаризованной системы, `anewm` – новый макрос `[y]=newm(t,xe,ue)`, который вычисляет поле значений линеаризованной системы (`newm(t,xe,ue)=0`). Применительно к исследуемой системе:

```
[A,B]=tangent('competition', x_e',u_e);
```

Зададим линеаризованную систему при начальном состоянии x_0:

```
C=[1,0];
x0=[1;1];
Sys=syslin('c',A,B,C,0,x0);
```

Наблюдатель для системы `Sys`:

```
Obs=observer(Sys);
```

Установим начальное состояние наблюдателя:

```
xch0=[1.5;1.5];
Obs('X0')=xch0;
```

Теперь отобразим состояния системы `Sys`. В качестве входного воздействия выбрана функция $u(t) = 100\sin(t)$:

```
Sys2=syslin('c',A,B,eye(2,2),[0;0],x0);
deff('ut=u(t)','ut=100*sin(t)');
```

Для расчета переходного процесса используется функция `csim`, входными параметрами которой являются строка-название функции Scilab, задающая входное воздействие, вектор моментов времени и имя исследуемой линейной динамической системы:

```
T=0:0.1:15;
x=csim(u,T,Sys2);
```

Зададим систему-наблюдатель $u \to \hat{x}$, промоделируем системы и отобразим процессы в наблюдателе и наблюдаемой системе на одном графике:

```
SObs=Obs*[1;Sys];
xhat=csim(u,T,SObs);
plot2d1("onn",T',[x(2,:);xhat(2,:)]')
```

Результат показан на рис. 4.2.

Для непрерывных систем временные характеристики могут быть получены при помощи функции `csim`, как показано выше. Моделирование дискретных систем (3) выполняется непосредственным рекурсивным решением матрично-векторных уравнений. Для этих целей используются встроенные функции `ltitr` и `rtitr`. Функция `ltitr` вычисляет x при заданном u, или при заданных u и x_0. Функцией `dsimul` может быть вычислен выходной сигнал $y = Cx + Du$. Рассмотрим дискретную систему с переходной матрицей $H(z) = \dfrac{N(z)}{D(z)}$, где $D(z)$ – квадратная неособая матрица многочленов размерностью $r \times r$, а $N(z)$ – матрица многочленов размерностью $r \times p$. Запишем:

$$D(z) = D_0 + D_1 z + ... + D_c z^d,$$
$$N(z) = N_0 + N_1 z + ... + N_n z^n.$$

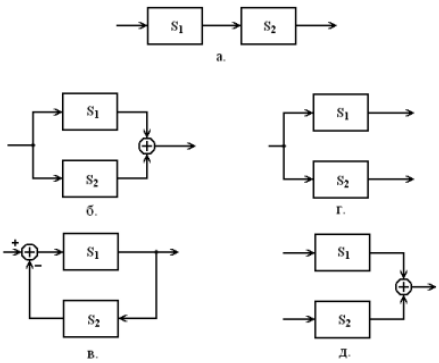

Рис. 4.1. Соединения линейных систем

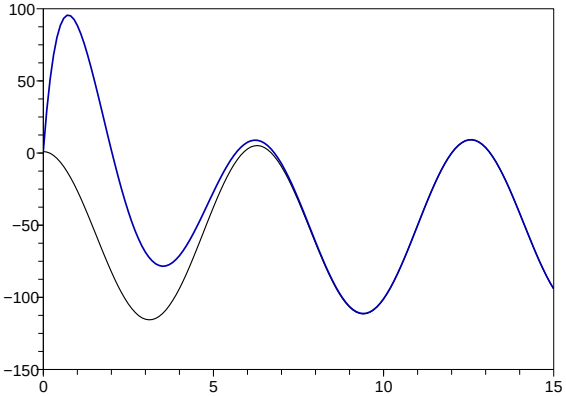

Рис. 4.2. Процессы в исходной системе и наблюдателе

Выход y системы, соответствующий входу u, может быть вычислен как решение рекурсивного уравнения:

$$D_d y(t+d) + D_{d-1} y(t+d-1) + ... + D_0 y(t) =$$
$$= N_n u(t+n) + N_{n-1} u(t+n-1) + ... + N_0 u(t).$$

Численное решение этого уравнения выполняется функцией `rtitr`, которая для заданного набора $u(k)$, $u(k+1), \ldots, u(k+T)$ возвращает значения $y(k)$, $y(k+1)$, $\ldots, y(k+T+d-n)$. В качестве примера рассмотрим:

```
N=[1;z;z^2];
D=[z^2+1,z-1,z;z+1,z^2-2,-z;z-3,z+4,z^2+5];
u=1:5;
y=rtitr(N,D,u)
```

Получим:

```
y  =
    0.   - 1.   - 3.     3.      22.
    0.     2.     5.     8.    - 5.
    1.     2.   - 3.   - 19.   - 20.
```

Рассмотрим систему (4). Преобразуем ее в дискретную систему с периодом дискретности 0.1 командой `Sys=dscr(Sys,0.1);`. Для расчета процессов в системе используем функцию `flts`. Выполним последовательность команд:

```
u=1:15;
y=flts(u,Sys)
plot(u,y,"+"); xgrid();
```

Результатом будет:

```
y  =
  column 1 to 8
 1. 0.9701592 0.9106599 0.8215116 0.7027240 0.5543068
 0.3762696 0.1686222
  column  9 to 15
 -0.0686260 -0.3354652 -0.6318859 -0.9578785
 -1.3134334 -1.698541 -2.1131918
```

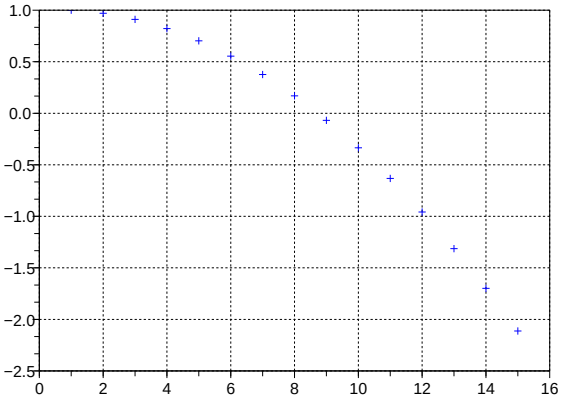

Рис. 4.3. Моделирование дискретной системы

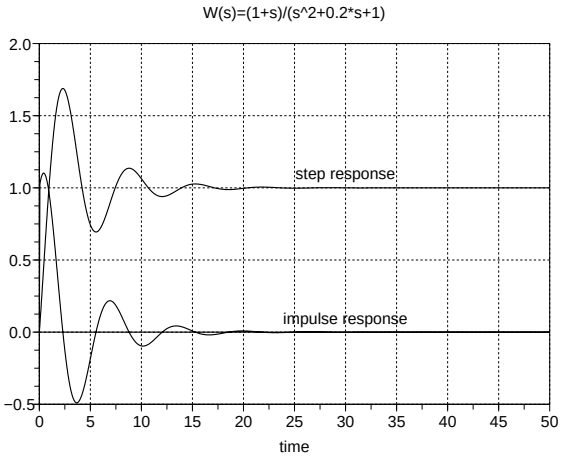

Рис. 4.4. Переходная и весовая функции непрерывной системы

Полученный график решения приведен на рис. 4.3.

Для непрерывных систем переходные характеристики при действии на систему ступенчатой функции и единичного импульса можно получить с помощью функции `csim`. Рассмотрим пример нахождения этих характеристик для динамической системы, заданной передаточной функцией:

$$W(s) = \frac{1+s}{s^2+0.5s+1}.$$

Зададим эту систему при помощи функции `syslin`, а затем вычислим ее реакцию на входные воздействия в виде единичного импульса и в виде ступенчатой функции. Для этого выполним следующую последовательность команд:

```
Sys=syslin("c", 1+%s,%s^2+0.5*%s+1)
T=0:0.05:50;
y1=csim('impulse', T, Sys);
y2=csim('step', T, Sys);
plot2d([T,T],[y1,y2]); xgrid();
xstring(30,1.1,'step response');
xstring(30,0.1,'impulse response');
xtitle('W(s)=(1+s)/(s^2+0.2*s+1)','time');
```

Результат показан на рис. 4.4.

Глава 5

Линейные матричные неравенства в среде SCILAB

Многие задачи анализа и синтеза систем управления могут быть сформулированы следующим образом [5,10]:

$$\sum : \begin{cases} \min\ f(X_1, ..., X_M) \\ \begin{cases} G_i(X_1, ..., X_M) = 0, & i = 1, 2, ..., p, \\ H_i(X_1, ..., X_M) \geq 0, & i = 1, 2, ..., p, \end{cases} \end{cases} \tag{1}$$

где $X_1, ..., X_M$ – неизвестные вещественные матрицы, f – вещественная линейная скалярная *целевая функция*, G_i – *линейное матричное равенство* (ЛМР), H_i – *линейное матричное неравенство* (ЛМН). Для решения задач типа (1) в SCILAB используется тулбокс LMITOOL [5,10].

5.1 Функция `lmisolver`

Основная функция этого тулбокса – `lmisolver`. Данная функция вычисляет решение $X_1, ..., X_M$ выражения (1) при заданных функциях f, G_i и H_i как функцию с неизвестными матрицами и позволяет задать приблизительные начальные значения искомых матриц. Рассмотрим синтаксис функции `lmisolver`:

[XLISTF[,OPT]] = lmisolver(XLIST0,EVALFUNC[,options])

`XLIST0` – список включающий в себя матрицы и/или списки матриц. Он содержит начальные приближенные значения неизвестных

матриц, i-й элемент XLIST0 в общем случае соответствует приблизительному значению марицы X_i. Значения матриц в XLIST0 используются как начальные значения алгоритма оптимизации. Размер и структура XLIST0 определяют размер выходного параметра XLISTF.

EVALFUNC – функция, определенная пользователем. Эта функция вычисляет ЛМУ, ЛМН и целевую функцию. Синтаксис EVALFUNC должен быть следующим:

[LME,LMI,OBJ]=EVALFUNC(XLIST),

где XLIST – список, идентичный с XLIST0 по размеру и структуре; LME – список матриц, содержащих значения функции ЛМР G_i для значений X в XLIST; LMI – список матриц, содержащих значения функции ЛМИ H_i для значений X в XLIST; OBJ – список матриц, содержащих значения целевой функции f для значений X в XLIST; options – вектор 5×1, содержащий параметры оптимизации Mbound, abstol, nu, maxiters, и reltol. Этот параметр необязателен. Если он опущен, используюся параметры, заданные по умолчанию; XLISTF – список, такой же по структуре и размеру, как и XLIST0, содержащий решения задачи (1) (оптимальные значения неизвестных матриц); OPT – скалярная величина, равная значению целевой функции f.

5.2 Примеры использования функции lmisolver

Найдем диагональную матрицу A такую, что

$$A_1^{\mathrm{T}} X + X A_1 + Q_1 < 0,$$
$$A_2^{\mathrm{T}} X + X A_2 + Q_2 < 0;$$

а след матрицы максимизирован. Для этого выполним следующие операторы:

```
n=2;
A1=rand(n,n);
A2=rand(n,n);
Xs=diag(1:n);Q1=-(A1'*Xs+Xs*A1+0.1*eye());
```

```
Q2=-(A2'*Xs+Xs*A2+0.2*eye());
deff('[LME,LMI,OBJ]=evalf(Xlist)','X=Xlist(1),...
    LME=X-diag(diag(X));...
    LMI=list(-(A1''*X+X*A1+Q1),-(A2''*X+X*A2+Q2)),...
    OBJ= -sum(diag(X))  ');
X=lmisolver(list(zeros(A1)),evalf);
X=X(1)
```

После вызова функции `lmisolver` в командном окне последовательно выводятся шаги ее исполнения. Для данного примера они выглядят следующим образом:

```
Construction of canonical representation
 Basis Construction
      FEASIBILITY PHASE.
    primal obj.  dual obj.  dual. gap
    1.07e+000   -1.98e-001  1.27e+000
   -8.01e-002   -1.69e-001  8.91e-002
 Target value reached
 feasible solution found
       OPTIMIZATION PHASE.
    primal obj.  dual obj.  dual. gap
   -2.92e+000   -3.67e+000  7.44e-001
   -3.09e+000   -3.18e+000  8.57e-002
   -3.10e+000   -3.10e+000  7.08e-003
   -3.10e+000   -3.10e+000  6.85e-004
   -3.10e+000   -3.10e+000  6.81e-005
   -3.10e+000   -3.10e+000  1.09e-005
   -3.10e+000   -3.10e+000  1.02e-006
   -3.10e+000   -3.10e+000  1.37e-007
   -3.10e+000   -3.10e+000  7.89e-009
 optimal solution found
```

Отсюда видно, что функция выполняется в два этапа: первый этап – *feasibility phase*, здесь проверяется возможность достижения искомого значения и осуществляется поиск возможного решения, в конце этого этапа выдается сообщение о возможности достижения искомой величины или о том, что искомая величина не может быть достигнута; на втором этапе – *optimization phase* – осуществляется

поиск оптимального решения. Полученная матрица выглядит следующим образом:

```
X  =
   1.0635042    0.
   0.           2.0784841
```

Рассмотрим линейную систему:

$$\dot{x} = Ax + Bu$$

Имеется стабилизирующая обратная связь K такая, что для каждого начального состояния $x(0)$, $\|x(0)\| \leq 1$ управление $\|u(t)\|$ для любых $t \geq 0$ тогда и только тогда, когда существуют матрицы Q и Y, удовлетворяющие равенству:

$$Q - Q^{\mathrm{T}} = 0$$

и неравенствам:

$$\begin{aligned} & Q \geq 0 \\ & -AQ - QA^{\mathrm{T}} - BY - Y^{\mathrm{T}}B^{\mathrm{T}} > 0 \\ & \begin{bmatrix} u_{max}^2 I & Y \\ Y^{\mathrm{T}} & Q \end{bmatrix} \geq 0 \end{aligned}$$

Таким образом, $K = YQ^{-1}$.

Для решения этой задачи при помощи функции `lmisolver` необходимо определить функцию:

```
function [LME,LMI,OBJ]=sf_sat_eval(XLIST)
[Q,Y]=XLIST(:)
LME=Q-Q'
LMI=list(-A*Q-Q*A'-B*Y-Y'*B',...
[umax^2*eye(Y*Y'),Y;Y',Q],Q-eye())
OBJ=[]
endfunction
```

При заданных заранее значениях A, B и u_{max} вызов функции `lmisolver` будет выглядеть следующим образом:

```
Q_init=zeros(A);
Y_init=zeros(B');
XLIST0=list(Q_init,Y_init);
XLIST=lmisolver(XLIST0,sf_sat_eval);
```

5.3 Признак разрешимости неравенства Ляпунова

При исследовании систем иногда бывает необходимо найти (или доказать невозможность нахождения) матрицу X, удовлетворяющую неравенствам:

$$E^{\mathrm{T}}X = X^{\mathrm{T}}E \geq 0$$
$$A^{\mathrm{T}}X + X^{\mathrm{T}}A + I \leq 0$$

При решении данной задачи важную роль играют функции ЛМН. Зададим оценочную функцию:

```
function [LME,LMI,OBJ]=dscr_lyap_eval(XLIST)
X=XLIST(:)
LME=E'*X-X'*E
LMI=list(-A'*X-X'*A-eye(),E'*X)
OBJ=[]
endfunction
```

Тогда решение задачи выглядит следующим образом (матрицы E и A считаем заданными заранее):

```
XLIST0=list(zeros(A))
XLISTF=lmisolver(XLIST0,dscr_lyap_eval)
X=XLISTF(:)
```

5.4 Функция `lmitool`

Назначение функции `lmitool` – автоматизировать большинство шагов, необходимых для вызова функции `lmisolver`, в частности она генерирует файл с расширением `sci`, который содержит решающую функцию, целевую функцию или по меньшей мере их скелет. Функция `lmitool` может быть вызвана с одним параметром, с тремя параметрами и без параметров. При вызове с тремя параметрами синтаксис следующий:
`txt=lmitool(probname,varlist,datalist)`
где `probname` – строка, созержащая имя задачи; `xlist` – строка, содержащая имена неизвестных матриц (имена разделены запятой,

если их больше чем одно); `dlist` – строка, содержащая имена матриц данных (имена разделены запятой, если их больше чем одно); `txt` – строка, содержащая информацию о том, что необходимо сделать пользователю для дальнейшего решения задачи.

При вызове функции `lmitool` таким образом, в текущей рабочей директории генерируется файл с расширением `sci` и с именем равным параметру `probname`. В этом файле содержится скелет решающей функции и соответствующей целевой функции.

Выполним команду:

```
txt=lmitool('sfsat','Q,Y','A,B,umax')
```

В командном окне появится сообщение:

```
 txt  =
!    To solve your problem, you need to       !
!1- edit file C:\Scilab_book\sfsat.sci         !
!2- load (and compile) your functions:         !
!   getf('C:\Scilab_book\sfsat.sci')           !
!3- Define A,B,umax and call sfsat function:   !
!  [Q,Y]=sfsat(A,B,umax)                       !
!To check the result, use                      !
![LME,LMI,OBJ]=sfsat_eval(list(Q,Y))          !
```

В этом сообщении выведена инструкция, в которой пользователю предлагается редактировать полученый файл, затем загрузить и скомпилировать его в Scilab, после чего задать данные, необходимые для решения задачи, и вызвать соответствующую функцию для получения результата.

Текст сгенерированного файла приведен ниже:

```
function [Q,Y]=sfsat(A,B,umax)
 // Generated by lmitool on
   Mbound = 1e3;
   abstol = 1e-10;
   nu = 10;
   maxiters = 100;
   reltol = 1e-10;
   options=[Mbound,abstol,nu,maxiters,reltol];
```

```
//////////DEFINE INITIAL GUESS AND PRELIMINARY
////////// CALCULATIONS BELOW
Q_init=...
Y_init=...
//////////

XLIST0=list(Q_init,Y_init)
XLIST=lmisolver(XLIST0,sfsat_eval,options)
[Q,Y]=XLIST(:)

//////////EVALUATION FUNCTION/////////////////
function [LME,LMI,OBJ]=sfsat_eval(XLIST)
[Q,Y]=XLIST(:)

/////////////////DEFINE LME, LMI and OBJ BELOW
 LME=...
 LMI=...
 OBJ=...
```

При вызове функции `lmitool` с одним параметром или без параметров синтаксис следующий:

`txt=lmitool(file)`

`txt=lmitool()`

где `file` – строка, содержащая название уже имеющегося файла с расширением `sci`, сгенерированного функцией `lmitool`.

При таком вызове функция полностью интерактивна, при помощи последовательности диалоговых окон пользователь может полностью описать задачу.

Рассмотрим задачу вычисления оценки:

$$y = Hx + Vw,$$

где x – неизвестная оцениваемая величина, y – известный параметр, а w – гауссов вектор с единичной дисперсией и нулевым средним,

$$H \in C_0\{H(1), ..., H(N)\}, \qquad V \in C_0\{V(1), ..., V(N)\}.$$

$H(i)$ и $V(i)$, $i = 1, ..., N$ – заданные матрицы.

Необходимо найти L, позволяющее вычислить оценку:

$$\hat{x} = Ly$$

Данная задача может быть сформулирована в виде (1):

$$I - LH(i) = 0, \quad i = 1, ..., N;$$
$$X(i) - X(i)^T = 0, \quad i = 1, ..., N;$$

и

$$\begin{bmatrix} I & (L(i)V(i))^T \\ L(i)V(i) & X(i) \end{bmatrix} \geq 0, \quad i = 1, ..., N;$$
$$\gamma - Trace(X(i)) \geq 0 \quad , i = 1, ..., N.$$

Вызовем функцию `lmitool`:

```
lmitool()
```

Результатом будет последовательность диалоговых окон, приведенных на рис. 5.1 – 5.5.

Рис. 5.1. Интерактивная функция `lmitool`

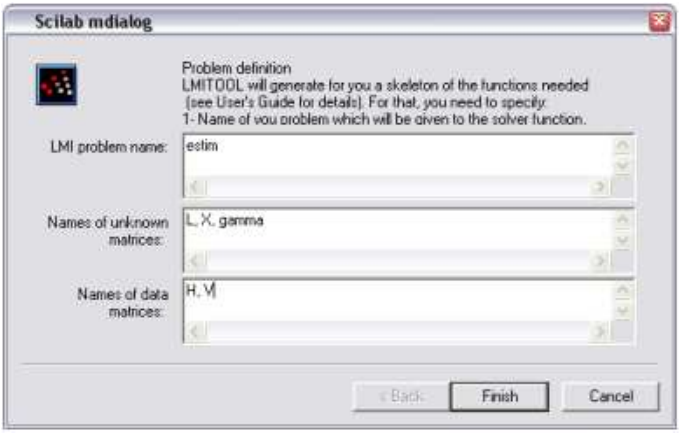

Рис. 5.2. Диалоговое окно для задания имени задачи и имен используемых переменных

Рис. 5.3. Структура полученной функции, отредактированной для решения поставленной задачи

Рис. 5.4. Диалоговое окно сохранения функции

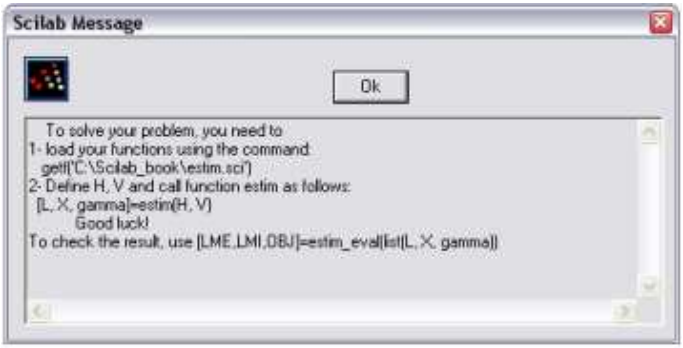

Рис. 5.5. Диалоговое окно с дальнейшими инструкциями для пользователя

Глава 6

Функции обработки сигналов

В среде Scilab имеется много функций обработки сигналов. Ниже будут рассмотрены основные из них, а именно отображение сигнала, расчет фильтров и оценка спектра сигнала [5].

6.1 Фильтрация и расчет фильтров

Линейный стационарный фильтр может быть рассчитан операцией свертки. Рассмотрим дискретный сигнал $x(n)$ и дискретный фильтр $h(n)$, тогда результат свертки $y(n)$ определяется как:

$$y(n) = \sum_{k=-\infty}^{\infty} h(n-k)x(k)$$

Свертка осуществляется функцией `convol`:

```
x=1:10;
h=ones(1,2);
y=convol(x,h)
```

С помощью функции `ones` создается матрица, состоящая из единиц, параметры этой функции – количество строк и столбцов создаваемой матрицы (в данном примере одна строка и два столбца). Результат свертки:

```
 y  =
    1.    3.    5.    7.    9.    11.    13.    15.    17.    19.    10
```

Фильтрацию дискретных сигналов линейных систем, созданных функцией `syslin`, можно выполнить функцией `flts`. Рассмотрим пример фильтра нижних частот:

```
f=iir(3,'lp','butt',[.1 0],[0 0])
```

Функцией `irr` задается фильтр с бесконечной импульсной характеристикой (БИХ-фильтр). Первый параметр функции – порядок фильтра, второй – строка, указывающая тип фильтра (`lp` - фильтр низких частот), третий аргумент указывает схему расчета фильтра (`butt` - фильтр Баттерворта), далее вектором из двух элементов задается частота среза (для фильтров низких и высоких частот используется только первый элемент вектора), последний аргумент – вектор из двух элементов, значения ошибки (используется только при фильтрах Чебышева и Кауэра). Полученный фильтр:

```
f  =
                                        2            3
    0.0180989 + 0.0542968z + 0.0542968z + 0.0180989z
    ------------------------------------------------
                                          2   3
      - 0.2780599 + 1.1828933z - 1.7600419z + z
```

Далее необходимо определить фильтр как линейную систему и задать входной сигнал:

```
f=syslin('d',f);
t=1:150;
x1=sin(2*%pi*t/20);
x2=sin(2*%pi*t/4);
x=x1+x2;
```

Проведем фильтрацию сигнала и построим соответствующие графики:

```
y=flts(x,f);
xsetech([0,0,1,.5])
plot(x)
xtitle('Signal input','time','amplitude')
xsetech([0,0.5,1,.5])
plot(y)
xtitle('Filtered signal','time','amplitude')
```

Функция `flts` позволяет получить временную характеристику дискретной системы `f` с входным сигналом `x`. Результат показан на рис. 6.1.

Расчет фильтров с конечной импульсной характеристикой (КИХ-фильтр):

$$H(z) = \sum_{k=0}^{N} h_k z^{-k}$$

может быть выполнен в Scilab функцией `wfir`.

```
[wft,wfm,fr]=wfir('lp',35,[.15 0],'kr',[2.8 0]);
xsetech([0,0,1,1/3])
plot2d(fr,20*log10(wfm))
xtitle('Low-Pass filter with Kaiser window',...
 'freq (Hz)','Amplitude (dB)')
[wft,wfm,fr]=wfir('sb',111,[.2 .4],'hm',[0 0]);
xsetech([0,1/3,1,1/3])
plot2d(fr,20*log10(wfm))
xtitle('Stop Band filter with Hamming window',...
 'freq (Hz)', 'Amplitude (dB)')
[wft,wfm,fr]=wfir('bp',55,[.20 .340],'ch',[.005 -1]);
xsetech([0,2/3,1,1/3])
plot2d(fr,20*log10(wfm))
xtitle('Band Pass filter with Chebyshev window',...
 'freq (Hz)', 'Amplitude (dB)')
```

Было создано три КИХ-фильтра: фильтр нижних частот с окном Кайзера, режекторный фильтр с окном Хэмминга и полосовой фильтр Чебышева. Первый аргумент функции – строка, в которой указан тип фильтра, второй аргумент – порядок фильтра (целое число), третий аргумент – вектор из двух элементов, указывающий промежуток срезаемых частот, четвертый аргумент – строка, в которой указывается тип окна, пятый аргумент – вектор из двух элементов, задающий параметры окна. Выходные параметры: `fr` – сетка частот, `wfm` – частотная характеристика фильтра (связанная с `fr`), `wft` – коэффициенты фильтра. Результат показан на рис. 6.2.

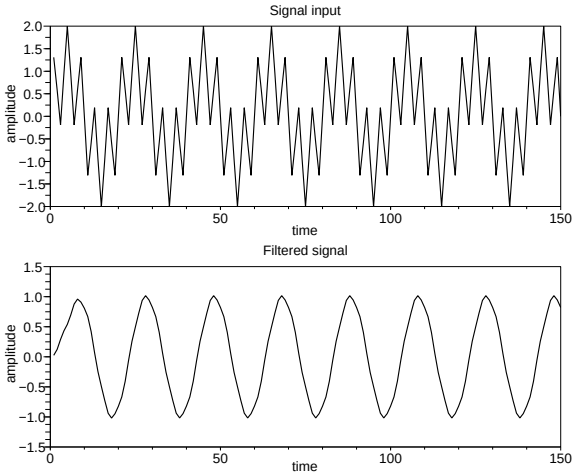

Рис. 6.1. Фильтрация сигнала при помощи функции `flts`

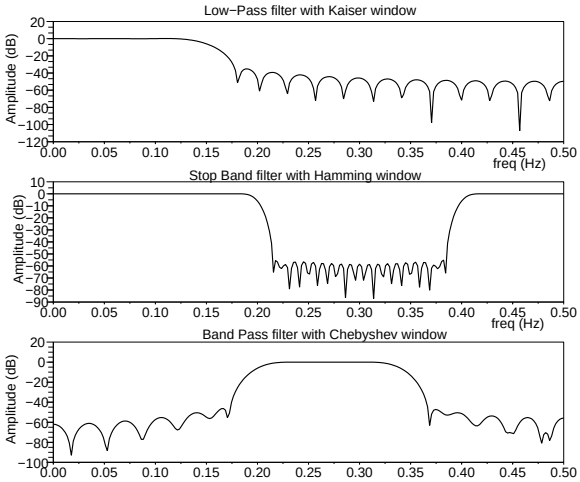

Рис. 6.2. АЧХ КИХ-фильтров, полученных при помощи функции `wfir`

При вызове этой функции без параметров параметры фильтра вводятся пользователем интерактивно при помощи последовательности диалоговых окон. Для расчета КИХ-фильтров, оптимизированных по минимаксному критерию, используется функция `eqfir`. Эта функция выдает фильтр с лучшими характеристиками, чем функция `wfir`, а также может быть использована для расчета многополосного фильтра. Рассмотрим пример, позволяющий сравнить фильтры, рассчитанные этими двумя функциями:

```
[wft,wfm,fr]=wfir('lp',45,[.23 0],'kr',[5.6 0]);
bedge=[0 .2;.2875 .5];
des=[1 0];
wate=[.025 1];
hn=eqfir(45,bedge,des,wate);
xsetech([0,0,1,.5])
plot2d(fr,20*log10(wfm)), xgrid()
xtitle('Low-Pass filter using wfir',...
 'freq (Hz)', 'Amplitude (dB)')
xsetech([0,.5,1,.5])
[hm,fr]=frmag(hn,256);
plot2d(fr,20*log10(hm)), xgrid()
xtitle('Low-Pass filter using eqfir',...
 'freq (Hz)', 'Amplitude (dB)')
```

Первый аргумент функции `eqfir` – целое число, задающее длину фильтра, `bedge` – матрица размером $M \times 2$, определяющая границы M полос пропускания фильтра, `des` – вектор длиной M, задающий желаемое значение для каждой полосы частот, `wate` – вектор длиной M, задающий допустимую величину ошибки для каждой полосы частот. Функцией `frmag` были получены значения частоты `hm` в точках `fr`. Входные параметры функции – вектор коэффициентов фильтра и число точек частотной характеристики. Результат показан на рис. 6.3.

Ниже приведен пример многополосного фильтра, полученного при помощи функции `eqfir`:

```
bedge=[0 .1;.15 .2;.25 .3;.35 .4;.45 .5;];
des=[0 1 0 .5 0];
wate=[1 .025 1 1 1];
```

```
hn=eqfir(45,bedge,des,wate);
[hm,fr]=frmag(hn,256);
plot2d(fr,hm), xgrid()
xtitle('5-band filter', 'freq (Hz)', 'Amplitude (dB)')
```

Результат показан на рис. 6.4. Для расчета фильтров с бесконечной импульсной характеристикой, помимо функции `iir`, может быть использована функция `eqiir`. Основное различие между ними в том, что в функции `eqiir` порядок фильтра не задается. Произведем расчет эллиптического фильтра низких частот, полосового фильтра Чебышева и режекторного фильтра Баттерворта:

```
[c,g]=eqiir('lp','ellip',2*%pi*[.1 .2],.05,.025);
[hf1,fr1]=frmag(g*prod(c(2))/prod(c(3)),256);
[c,g]=eqiir('hp','cheb2',2*%pi*[.3 .4],.05,.025);
[hf2,fr2]=frmag(g*prod(c(2))/prod(c(3)),256);
[c,g]=eqiir('sb','butt',2*%pi*[.1 .2 .3 .4],.05,.025);
[hf3,fr3]=frmag(g*prod(c(2))/prod(c(3)),256);
xsetech([0,0,1,1/3])
plot2d(fr1,hf1), xgrid()
xtitle('Elliptic low-pass filter', 'freq (Hz)',...
 'Amplitude (dB)')
xsetech([0,1/3,1,1/3])
plot2d(fr2,hf2), xgrid()
xtitle('Chebyshev band-pass filter', 'freq (Hz)',...
 'Amplitude (dB)')
xsetech([0,2/3,1,1/3])
plot2d(fr3,hf3), xgrid()
xtitle('Butterworth stop-band filter', 'freq (Hz)',...
 'Amplitude (dB)')
```

Функция `eqiir` имеет следующие параметры: первый – строка, указывающая тип фильтра, второй – строка, определяющая схему расчета фильтра, третий – вектор из четырех элементов, задающий частоты среза (для фильтров низких и высоких частот указываются только первые два элемента вектора), четвертый – пульсация в полосе пропускания, пятый – величина пульсации в полосе непропускания. Функция возвращает вектор рациональных функций первого и второго порядка, дающих при перемножении передаточную функцию фильтра. Результат показан на рис. 6.5.

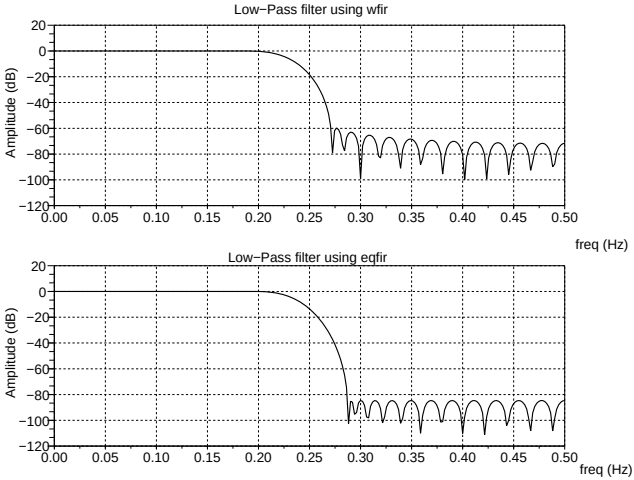

Рис. 6.3. Сравнение АЧХ фильтров, созданных функциями `wfir` и `eqfir`

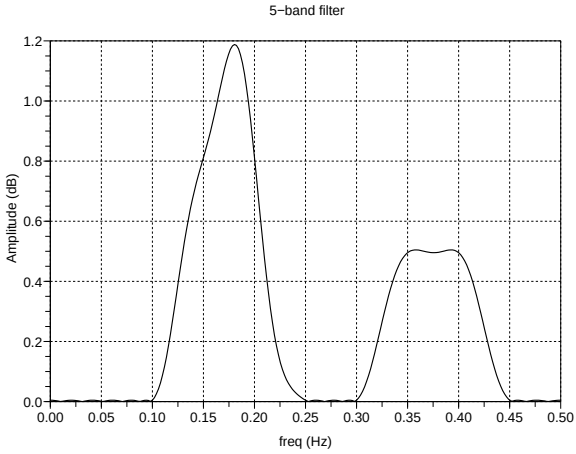

Рис. 6.4. АЧХ многополосного фильтра, полученного при помощи функции `eqfir`

6.2 Вычисление спектральной функции

Спектральная функция детерминированного конечного дискретного сигнала $x(n)$ определяется как квадрат модуля изображения Фурье этого сигнала:

$$S_x(\omega) = \frac{1}{N}\left|\sum_{n=0}^{N-1} x(n)e^{-j\omega n}\right|^2.$$

Рассмотрим несколько способов расчета спектра сигнала с использованием среды Scilab.

6.2.1 Вычисление спектра случайного сигнала методом периодограмм

Первый метод – *метод I периодограмм*. Периодограмма конечной последовательности определяется как:

$$I(\omega) = \frac{1}{U}\left|\sum_{n=0}^{N-1} \omega(n)x(n)e^{-j\omega n}\right|^2,$$

где $w(n)$ – вырезающая функция. Если для вычисления сигнала используются K сегментов длиной N, то периодограмма оценки спектра $\hat{S}_x$ является средним значением K периодограмм:

$$\hat{S}_x(\omega) = \frac{1}{K}\sum_{k=0}^{K-1} I_k,$$

Эти периодограммы усреднены и нормализованы для достижения требуемой оценки спектральной функции сигнала.

Вычисление спектральной функции методом периодограмм осуществляется функцией `pspect`.

В качестве примера рассмотрим случайный сигнал, полученный пропусканием белого шума с единичной дисперсией через фильтр низких частот.

```
rand('normal');
rand('seed',0);
```

```
x=rand(1:4096-33+1);
nf=33;
bedge=[0 .1;.125 .5];
des=[1 0];
wate=[1 1];
h=eqfir(nf,bedge,des,wate);
y=convol(h,x);
sm=pspect(100,200,'tr',y);
[hf,fr]=frmag(h,100);
xsetech([0 0 1 .5])
plot2d(fr,20*log10(hf)), xgrid();
xtitle('Squared magnitude of the filter response',...
      'freq (Hz)','magnitude (db)')
xsetech([0 .5 1 .5])
plot2d(fr,20*log10(sm(1:100))), xgrid();
xtitle('Estimate of the spectrum',...
      'freq (Hz)','magnitude (db)')
```

Для вызова функции `pspect` необходимо передать следующие параметры: сдвиг сегмента данных, длина каждого сегмента данных, тип окна (в данном примере используется треугольное окно) и вектора данных, задающих сигнал. В функцию можно передать вектор второго сигнала, тогда будет найдена оценка их взаимного спектра. Результат показан на рис. 6.6.

6.2.2 Вычисление спектра случайного сигнала методом корреляции

Второй метод, *метод корреляции*, заключается в вычислении преобразования Фурье усредненной оценки автокорреляционной функции. Для каждого N-точечного сегмента данных, $x_k(n)$, оценка $2M$ точек автокорреляционной функции вычисляется следующим образом:

$$\hat{R}_k(m) = \sum_{n=0}^{N-1-m} x(n+m)x^*(n)$$

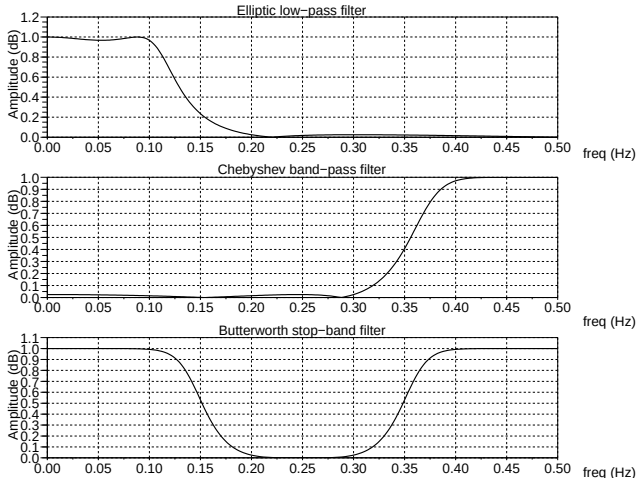

Рис. 6.5. АЧХ БИХ-фильтров, полученных при помощи функции `eqiir`

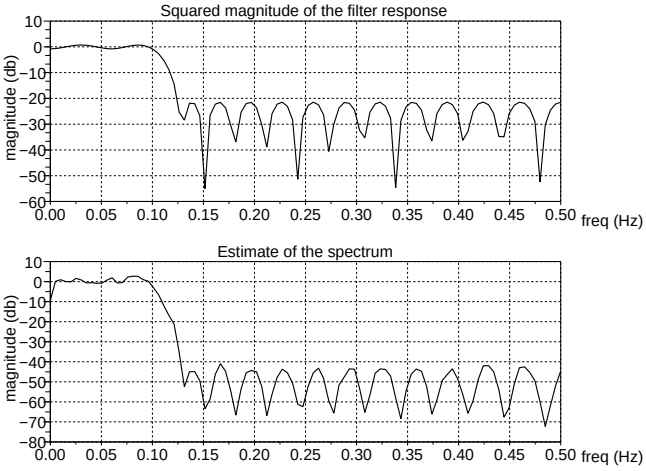

Рис. 6.6. Спектр сигнала, рассчитанный по `pspect`

для $m = 0, \pm 1, \pm 2, ..., \pm M$ оценка спектральной функции принимает вид:

$$\hat{S}_x(\omega) = F\left\{\tilde{R}_x(m)\omega(m)\right\},$$

где F – оператор преобразования Фурье, $\omega(m)$ – вырезающая функция, $\tilde{R}_x(m)$ – среднее значение K оценок:

$$\tilde{R}_x = \frac{1}{K}\sum_{k=1}^{K}\hat{R}_k.$$

Корреляционный метод оценки спектра реализован функцией `cspect`. Рассмотрим применение этой фунции, используя сигнал из предыдущего примера:

```
rand('normal');
rand('seed',0);
x=rand(1:4096-33+1);
nf=33;
bedge=[0 .1;.125 .5];
des=[1 0];
wate=[1 1];
h=eqfir(nf,bedge,des,wate);
y=convol(h,x);
sm=cspect(33,200,'ch',y,[.02,-1]);
[hf,fr]=frmag(h,100);
xsetech([0 0 1 .5])
plot2d(fr,20*log10(hf)), xgrid();
xtitle('Squared magnitude of the filter response',...
      'freq (Hz)','magnitude (db)')
xsetech([0 .5 1 .5])
plot2d(fr,20*log10(sm(1:100))), xgrid();
xtitle('Estimate of the spectrum','freq (Hz)',...
       'magnitude (db)')
```

Первые два параметра функции `pspect` – количество интервалов корреляции и количество преобразуемых точек, следующие параметры, как и в `pspect`, – выбор окна (в данном случае – окно Чебышева) и вектор, задающий сигнал. Последний аргумент – параметры выбранного окна. Результат показан на рис. 6.7.

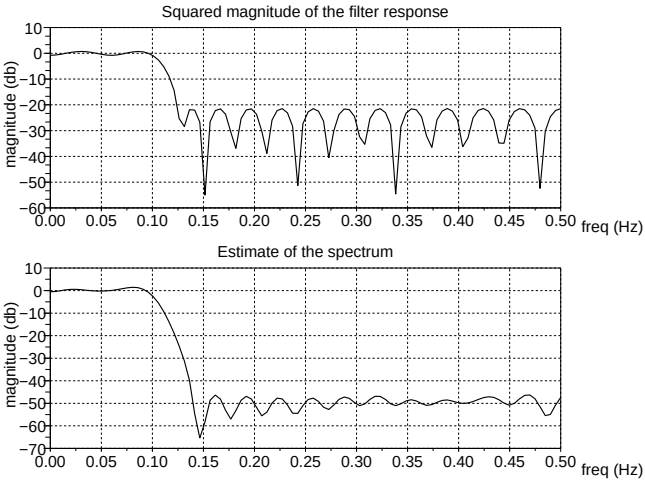

Рис. 6.7. Спектр сигнала, рассчитанный по `cspect`

6.2.3 Вычисление спектра квазипериодического сигнала преобразованием Фурье

Обратимся к задаче нахождения спектра сигнала по дискретной выборке на основе преобразования Фурье [1, 5]. Рассмотрим квазипериодический процесс, представляющий собой сумму гармоник с несоизмеримыми частотами:

$$y(t) = \sin(t) + 0.75\sin(5\pi^{-1}t).$$

Зададимся интервалом дискретности $T_0 = 0.05$ и числом точек отсчетов $N = 2^{10} = 1024$. Это соответствует длине реализации $T = 51.2$. Программа вычисления энергетического спектра представлена ниже:

```
clf();
dt=0.05;
Nt=2^10;
t=0:dt:(Nt-1)*dt;
y=sin(t)+0.75*sin(5/%pi*t);
plot2d(t,y)
xtitle('y(t)')
```

В этом фрагменте программы задается число точек для вычисления спектра и находится период дискретости. Для этого вводится массив момента отсчета `t` с постоянным шагом `dt`, затем формируется подлежащий дальнейшей обработке массив `y`, соответствующий значениям функции $y(t)$ для моментов отсчета `t`. Оператором `plot2d(t,y)` выводится график этой функции. После этого выполняется *дискретное преобразование Фурье* (ДПФ) с помощью процедуры `fft` и вычисляются энергетические спектры сигнала:

```
Y=fft(y,-1);
PY=abs(Y).^2/Nt;
wn2=2*%pi/dt;
dw=wn2/Nt;
```

Построим график спектра процесса в диапазоне частот $[0, 5]$ с$^{-1}$:

```
w=0:dw:5;
lw=length(w);
```

```
xset('window',1)
plot2d(w,PY(1:lw))
xtitle('S(w)')
```

Результаты приведены на рис. 6.8

Глава 7

Решение задач оптимизации

Оптимизация как выбор наилучшего варианта среди некоторого множества подразумевает наличие правил предпочтения одного варианта другому – критерия оптимальности. В основе построения такого правила лежит целевая функция (функция качества), аргументами которой являются управляемые параметры – внутренние параметры x, которые можно изменять на данном этапе проектирования. Целевую функция обозначим через $f(x)$, область ее определения – через X_D. Если область определения дискретная, то задача относится к задаче дискретного (в частном случае целочисленного) программирования. Множество $S_\epsilon(x)$ точек, которые находятся от точки x_0 на расстоянии, не превышающем заданное $\epsilon > 0$, называют ϵ-окрестностью точки x_0 :

$$S_\epsilon(x) = \{x : \|x - x_0\| \leq \epsilon\}.$$

Максимумом функции $f(x)$ называют ее значение $f(x^*)$, если существует число $\epsilon > 0$, такое, что для любой точки $x \in S_\epsilon(x^*)$ $(x \neq x^*)$ выполняется неравенство:

$$f(x) - f(x^*) < 0.$$

Точку x^* называют *локальным экстремумом* (экстремальной точкой). Если функция $f(x)$ имеет один экстремум, то она одноэкстремальна, и многоэкстремальна, если имеет более одного максимума

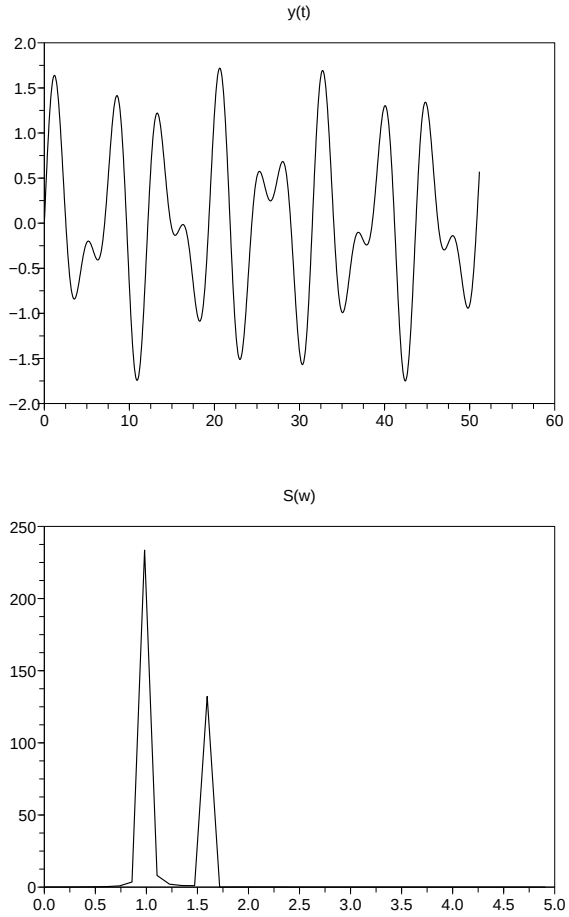

Рис. 6.8. Квазипериодический сигнал и его спектр

(миниума). Точка, в которой целевая функция имеет наибольшее (при решении задач максимизации) значение среди всех локальных экстремумов, называется точкой глобального экстремума. В задачах проектирования обычно имеются ограничения. Прямые ограничения описываются неравенствами

$$x_{d_i} < x_i < x_{u_i},$$

а область X_P в пространстве параметров, удовлетворяющую прямым ограничениям, называют допустимой областью:

$$X_P =^{\Delta} \{x : x \in X_D | x_{d_i} < x_i < x_{u_i}, i \in [1:n]\}, \quad n = dim X_D.$$

Кроме прямых ограничений есть также функциональные, имеющие вид равенств или неравенств:

$$\psi = 0, \quad \varphi > 0.$$

Наличие ограничений приводит к задаче условной оптимизации, в результате решения которой находится условный экстремум. К задачам с отсутствие ограничений относят задачи безусловной оптимизации, найденные при этом экстремумы называются безусловными.

В качестве примера рассмотрим задачу нахождения экстремума функции двух переменных и ее решения с помощью пакета Scilab.

```
xref=[1;2;3];x0=[1;-1;1]
  deff('[f,g,ind]=cost(x,ind)',...
       'f=0.5*norm(x-xref)^2,g=x-xref');
  [f,xopt]=optim(cost,x0)      //Simplest call

 xopt  =
    1.
    2.
    3.
 f  =
    0.
```

Функция `optim` имеет следующие основные параметры: `[f [,xopt [,gradopt [,work]]]]=optim(costf ,x0 [,algo] [,df0 [,mem]]`

[,work] [,<params>] [,imp=iflag]) . Для использования функции optim необходимо определить *функцию потерь* (*целевую функцию*), которая имеет следующий вид: [f,g,ind]=costf(x,ind), где f и g – значения функции потерь и ее градиента, переменная ind используется как входной и выходной аргумент и определяет настройку вычисления функции потерь. Если ind равен 2, 3 или 4, то функция должна вычислять только f, g и f или только g. Если ind=0, то функция costf не вычисляет ничего и используется только для вывода.

x0 – вектор вещественных чисел, определяющий начальное значение оптимизируемой переменной;

f – значение функции потерь после оптимизации (f=costf(xopt));

xopt – найденное лучшее значение x;

algo – строка, с помощью которой задается алгоритм оптимизации ('qn' – квази-ньютоновский, 'gc' – сопряженных градиентов или 'nd' – без дифференцирования);

df0 – приблизительное убывание f после первой итерации (по умолчанию df0=1);

mem – количество переменных, используемых для аппроксимации гессиана, (для algo='gc' или 'nd'). По умолчанию – 6;

nap – максимальное количество вызовов фунции costf (по умолчанию – 100);

iter – максимальное количество итераций (по умолчанию – 100).

Для облегчения использования функции optim существует функция NDcost, служащая для вычисления градиента с использованием конечных разностей. Параметры этой функции: [f,g,ind]=NDcost(x,ind,fun,varargin)

x – вектор или матрица вещественных чисел;

ind – целочисленный параметр (см. optim);

fun – функция, имеющая следующий вид: F=fun(x,varargin), varargin может использоваться для передачи параметров p1,...pn;

f – значение критерия в точке x (см. optim);

g – значение градиента в точке x (см. optim).

Рассмотрим пример минимизации функции Розенброка. Представим ее в системе Scilab:

```
function f=rosenbrock(x,varargin)
   p=varargin(1)
   f=1+sum( p*(x(2:$)-x(1:$-1)^2)^2 + (1-x(2:$))^2)
endfunction
```

Теперь минимизируем данную функцию при помощи функции `optim`, передавая вместо функции `costf` список, содержащий обращение к функции `NDcost`, обращение к пользовательской функции `rosenbrock` и параметр `varargin`:

```
x0=[1;2;3;4];
[f,xopt,gopt]=optim(list(NDcost,rosenbrock,200),x0)
```

Получим следующие результат:

```
gopt  =
    0.0000002
    0.0000002
  - 1.414D-08
  - 5.112D-08
 xopt  =
    1.
    1.
    1.
    1.0000000
 f  =
    1.
```

```
// Модель линейного колебательного звена:
//
//   x''(t) + c x'(t) + k x(t) = 0 ,
//
// Перепишем ее в виде системы
// дифференциальных уравнений
// первого порядка, обозначив
// y(1) = x и y(2) = x':
//
//     dy1/dt = y(2)
//     dy2/dt = -c*y(2) -k*y(1).
//
```

```
// Пусть есть m измерений of x (соответствующих y(1))
// в разные моменты времени
// t_obs(1), ..., t_obs(m),
// обозначенные x_obs(1), ..., x_obs(m)
// и мы хотим идентифицировать параметры
// c и k минимизируя сумму квадратов ошибок
// между x_obs и y1(t_obs,p).
//
function dy = DEQ(t,y,p)
// Правые части нашей системы ОДУ
  c =p(1);k=p(2)
  dy=[y(2);-c*y(2)-k*y(1)]
endfunction

function y=uN(p, t, t0, y0)
   // Численное решение получим по ode.
      y = ode(y0,t0,t,list(DEQ,p))
endfunction

function r = cost_func(p, t_obs, x_obs, t0, y0)
   // Это минимизируемая функция,
   // являющаяся суммой квадратов ошибок
   // между значениями выхода модели
   // и результатами измерений
   sol = uN(p, t_obs, t0, y0)
   e = sol(1,:) - x_obs
   r = sum(e.*e)
endfunction

// Данные
y0 = [10;0]; t0 = 0; // Начальное значение y0
                     // для начального момента t0.
T = 30;  // Конечное время для измерений.

// Эдесь мы моделируем данные эксперимента
pe = [0.2;3];  // Точные значения параметров
m = 80;  t_obs = linspace(t0+2,T,m);
// Моменты наблюдения
```

```
sigma = 0.1;
y_exact = uN(pe, t_obs, t0, y0);
x_obs = y_exact(1,:) + ...
grand(1,m,"nor",0, sigma).*abs(y_exact(1,:));

// Начальное приближение для параметров
p0 = [0.5 ; 5];

// Значение функции потерь до оптимизации:
cost0 = cost_func(p0, t_obs, x_obs, t0, y0);
mprintf("\n\r The value of the cost function before
optimization = %g \n\r",cost0)
// Решение по optim
[costopt,popt]=optim(list(NDcost,cost_func,...
 t_obs, x_obs, t0, y0),p0,...
 'ar',40,40,1e-3);
mprintf("\n\r The value of the cost function after
 optimization  = %g",costopt)
mprintf("\n\r The identified values of the parameters:
 c = %g, k = %g \n\r", popt(1),popt(2))
// Вывод графика:
t = linspace(0,T,400);
y = uN(popt, t, t0, y0);
clf();
plot2d(t',y(1,:)',style=5)
plot2d(t_obs',x_obs(1,:)',style=-5)
legend(["model","measurements"]);
xtitle("Least square fit to identify ode parameters")
```

```
 The value of the cost function
           before optimization = 639.53
 The value of the cost function
           after optimization  = 5.65252
 The identified values of the parameters:
           c = 0.206094, k = 3.00132
```

Результат показан на рис. 7.1.

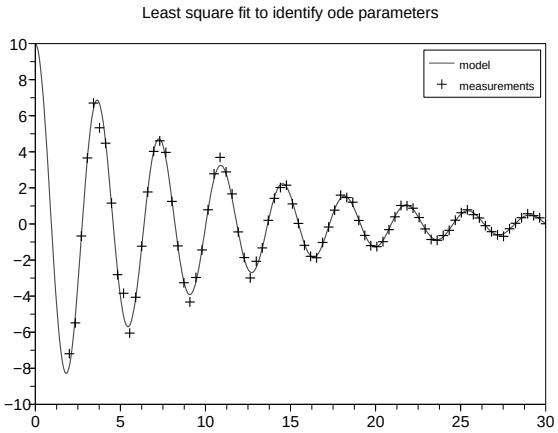

Рис. 7.1. Исходный процесс и результат идентификации

Литература

[1] *Андриевский Б.Р.*, Фрадков А.Л., Элементы математического моделирования в программных средах MATLAB 5 и Scilab: учебное пособие. СПб.: Наука, 2001.

[2] *Андриевский А.Б.*,Андриевский Б.Р., Фрадков А.Л., Использование системы SCILAB : практическое пособие. Санкт-Петербург: БГТУ, 2010.

[3] *Алексеев Е.Р.*, Чеснокова О.В., Рудченко Е.А. Scilab: Решение инженерных и математических задач. М: ALT Linux Бином. Лаборатория знаний. 2007. http://shop.altlinux.ru/index.php?productID=669

[4] *Introduction* to SCILAB. Consortium sCilab. Rocquencourt, France. 2010. [Online]. Available: http://www.scilab.org/resources/documentation/tutorials

[5] *Bunks C.*, Chancelier J.-P., Delebecque F., *et. al.* Engineering and Scientific Computing with Scilab, C. Gomez, Ed. Boston, Basel, Berlin: Birkhäuser, 1998.

[6] *Андриевский Б.Р.*, Фрадков А.Л. Избранные главы теории автоматического управления с примерами на языке MATLAB. СПб.: Наука, 1999.

[7] *Nikoukhah R.*, Steer S. Scicos – A Dynamic System Builder and Simulator. User's Guide. Rocquencourt, France: INRIA, 1998. [Online]. Available: http://www.scicos.org/

[8] *Бесекерский В.А.*, Попов Е.П. Теория систем автоматического управления. изд. 4. СПб: Профессия, 2003.

[9] *Уткин В.И.* Скользящие режимы в задачах оптимизации и управления. М.: Наука, 1981.

[10] *Nikoukhah R.*, Delebecquey F., Ghaouiz L. El. LMITOOL: a Package for LMI Optimization in Scilab. User's Guide. Rocquencourt, France: INRIA, 1998. [Online]. Available: http://www.scilab.org/doc/lmidoc/lmi.pdf

В 2009 году Университет стал победителем многоэтапного конкурса, в результате которого определены 12 ведущих университетов России, которым присвоена категория "Национальный исследовательский университет". Министерством образования и науки Российской Федерации была утверждена программа его развития на 2009-2018 годы. В 2011 году Университет получил наименование "Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики".

КАФЕДРА СИСТЕМ УПРАВЛЕНИЯ И ИНФОРМАТИКИ

Кафедра Систем Управления и Информатики (до 2001 г. кафедра Автоматики и Телемеханики) факультета Компьютерных Технологий и Управления была основана в 1945 г. на базе факультета Электроприборостроения ЛИТМО. На кафедру Автоматики и телемеханики ЛИТМО была возложена задача подготовки специалистов по автоматизации приборостроительной промышленности, автоматических систем управления, систем телемеханики и телеизмерений. Первый выпуск молодых инженеров состоялся в 1948 г. и составил 17 человек. Первым заведующим кафедры был крупный специалист в области систем телеизмерений, профессор Марк Львович Цуккерман.

В 1955 г. при кафедре образована научно-исследовательская лаборатории (НИЛ). В этот период основные направления научно-исследовательских работ представляли задачи автоматизации измерения и регистрации параметров кораблей во время их мореходных испытаний, а также стабилизации скорости и фазирования двигателей. Под научным руководством проф. М.Л. Цуккермана была налажена подготовка научных кадров высшей квалификации через систему аспирантуры.

С 1959 г. по 1970 кафедру возглавлял ученик М.Л. Цуккермана доцент Ефимий Аполлонович Танский. За время его руководства в научно-исследовательской работе на кафедре произошел заметный поворот к проблемам автоматизации оптико-механического приборостроения, что привело к длительному научно-техническому сотрудничеству кафедры с ЛОМО им. В.И. Ленина, в рамках которого для нужд оборонной техники была разработана целая гамма прецизионных фотоэлектрических следящих систем. В рамках научно-технического сотрудничества с НИИЭТУ кафедра приняла участие в разработке автоматической фототелеграфной аппаратуры, реализованной в виде комплекса "Газета-2".

С 1970 по 1990 г., за время руководства кафедрой известного в стране специалиста в области автоматизированного электропривода и фотоэлектрических следящих систем доктора технических наук, профессора Юрия Алексеевича Сабинина, заметно изменилась структура дисциплин и курсов, читаемых студентам кафедры. К традиционным курсам "Теория автоматического регулирования и следящие системы", "Теория автоматического управления, экстремальные и адаптивные системы", "Элементы автоматики" и "Телемеханика" были добавлены дисциплины: "Теоретические основы кибернетики", "Локальные системы управления", "САПР систем управления" и другие. Прикладные разработки кафедры были связаны с задачами адаптивной оптики для многоэлементных зеркал оптических телескопов и коррекции волнового фронта технологических лазеров; с задачами адаптивной радиооптики применительно к проблеме управления большими полноповоротными радиотелескопами; гребного электропривода и робототехнических систем, автоматического управления процессом мягкой посадки летательных аппаратов.

С 1990 г. научно-исследовательская работа кафедры велась по федеральным целевым программам и конкурсным проектам РФФИ, Минобразования и Администрации Санкт-Петербурга. С целью расширения исследований, проводимых по теории нелинейных и адаптивным систем, роботов и микропроцессорной техники, а также активизации подготовки кадров в 1994 г. образована научная Лаборатория Кибернетики и Систем управления (руководитель проф. И.В. Мирошник). С 1994 г. существенно расширились международные контакты кафедры, участие в самых престижных международных научных мероприятиях, организации конференций и симпозиумов. С 1998 г. на базе кафедры в университете ежегодно проводится Международная студенческая олимпиада по автоматическому управлению, а с 2009 года проводится Всероссийский Фестиваль Мехатроники и Робототехники.

В 2001 г. кафедра была переименована и получила название "Кафедра Систем управления и информатики". В 2010 г. кафедру возглавил доктор технических наук, профессор Бобцов Алексей Алексеевич, работающий в то время уже в должности декана факультета Компьютерных технологий и управления. Профессор Бобцов А.А. является председателем советов молодых ученых Санкт-Петербурга и Северо-Западного Федерального Округа. В настоящее время кафедра является одним из ведущих российских научных и образовательных центров, ориентированным на фундаментальные и прикладные исследования в области систем автоматического управления, робототехники и прикладной информатики, подготовку высококвалифицированных специалистов XXI столетия.

Андриевский А.Б., Андриевский Б.Р.,Капитонов А.А.,
Фрадков А.Л.

Решение инженерных задач в Scilab

Учебное пособие

В авторской редакции
Компьютерная верстка А.Б. Андриевский, А.А. Капитонов
Дизайн обложки и иллюстраций А.Б. Андриевский, А.А. Капитонов

Редакционно-издательский отдел НИУ ИТМО
Зав. РИО Н.Ф. Гусарова
Лицензия ИД № 00408 от 05.11.99
Подписано к печати 10.10.2013
Заказ № 3036
Тираж 100 экз.
Отпечатано на ризографе

Редакционно-издательский отдел
Санкт-Петербургского национального
исследовательского университета
информационных технологий, механики и оптики
197101, Санкт-Петербург, Кронверкский пр., 49