

Алексеев Е.Р., Чеснокова О.В

Scilab

теория и практика

Донецк – 2007

1. Пакет Scilab. Начало работы.

Scilab – это система компьютерной математики, которая предназначена для выполнения инженерных и научных вычислений, таких как:

- решение нелинейных уравнений и систем;
- решение задач линейной алгебры;
- решение задач оптимизации;
- дифференцирование и интегрирование;
- задачи обработки экспериментальных данных (интерполяция и аппроксимация, метод наименьших квадратов);
- решение обыкновенных дифференциальных уравнений и систем.

Кроме того, Scilab предоставляет широкие возможности по созданию и редактированию различных видов графиков и поверхностей.

Не смотря на то, что система Scilab содержит достаточное количество встроенных команд, операторов и функций, отличительная ее черта это гибкость. Пользователь может создать любую новую команду или функцию, а затем использовать ее наравне со встроенными. К тому же, система имеет достаточно мощный собственный язык программирования высокого уровня, что говорит о возможности решения новых задач.

1.1. Установка Scilab на ПК

Свободно распространяемую версию пакета вместе с полной документацией на английском языке можно получить на сайте программы www.scilab.org.

Существуют версии Scilab для операционных систем Windows и Linux. Они имеют некоторые отличия в названиях пунктов главного меню, но команды пакета в обеих версиях идентичны.

Рассмотрим особенности установки пакета для операционной системы Windows.

Для того чтобы установить четвертую версию пакета на ПК необходимо обратиться к выполняемому файлу `scilab-4.x`, после чего начнет свою работу **Мастера установки**. В первом окне **Мастера установки** нужно выбрать язык (английский или французский) и нажать кнопку **ОК** для продолжения инсталляции. Следующее окно является информационным. Пользователь получает сообщение о том, что на его компьютер будет установлена четвертая версия пакета Scilab и рекомендацию закрыть другие приложения перед установкой. Для перехода к третьему окну **Мастера установки** следует выбрать кнопку **Next**. В этом окне следует принять условия лицензионного соглашения (**I accept the agreement**) и нажать клавишу **Next** для продолжения. На следующем этапе пользователю

будет предложено выбрать путь для установки пакета. По умолчанию это папка `C:\Program Files\scilab-4.0-rc1`. Другой путь можно установить при помощи кнопки **Browse...** Кроме того, в этом окне выводится информация о количестве места на выбранном диске, требуемого для установки стандартного набора компонентов системы. Нажатие кнопки **Next** приведет к следующему шагу (рис. 1.1), где пользователю будет предложено выбрать один из типов инсталляции:

- инсталляция по умолчанию (**Installation Default**);
- полная инсталляция (**Full installation**);
- компактная инсталляция (**Compact installation**);
- инсталляция с выбором компонентов необходимых для установки (**Custom installation**).

Далее будет приведен список компонентов соответствующих выбранному типу установки. Переход к следующему окну **Мастера установки** осуществляется с помощью кнопки **Next**.

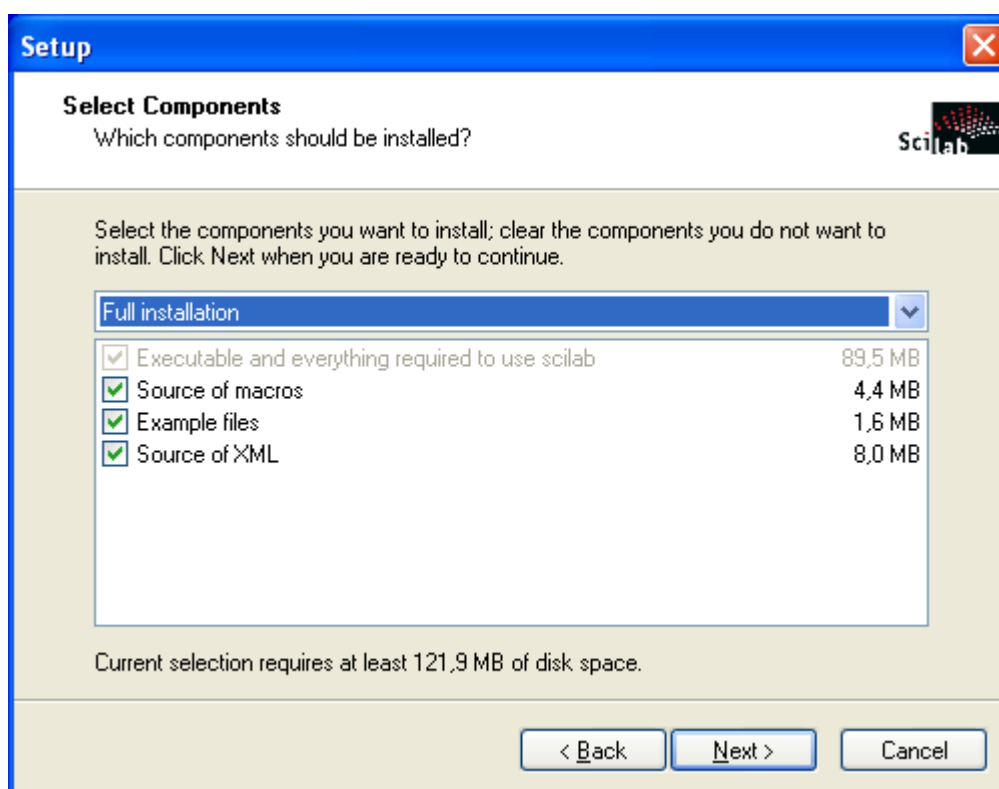


Рис. 1.1. Окно выбора компонентов для установки Scilab

Следующее окно мастера установки сообщает пользователю о том, что после инсталляции в меню **Пуск** будет создан ярлык, предназначенный для запуска Scilab. По умолчанию ему будет присвоено имя `scilab-4.x`. Изменить параметры ярлыка в меню **Пуск** можно при помощи кнопки **Browse...** Нажатие кнопки **Next** приведет к появлению следующего окна, в котором **Мастер установки** предложит список дополнительных задач, доступных после инсталляции. Для перехода к следующему шагу так же необходимо выбрать

Next.

Перед началом установки **Мастер** выдает краткий отчет о параметрах инсталляции. Выбор кнопки **Install** приведет к началу процесса инсталляции. Процесс установки Scilab заключается в копировании файлов системы на жесткий диск. Контролировать процесс инсталляции пользователь может с помощью, так называемого, линейного индикатора. Завершается процесс установки появлением информационного окна. Нажатие кнопки **Next** приведет к последнему шагу, где у пользователя будет возможность установить единственную опцию **Launch scilab**. Если этот параметр активен, то Scilab запустится сразу после нажатия кнопки **Finish**, иначе запуск можно произвести из главного меню (**Пуск\Программы\scilab-4.x**) или с помощью ярлыка на рабочем столе (рис. 1.2.):



Рис. 1.2. Ярлык для запуска Scilab

Перейдем к рассмотрению установки пакета для операционной системы Linux.

Scilab входит в состав таких дистрибутивов, как ALT Linux и Mandriva 2007. Для установки в другие дистрибутивы Linux необходимо загрузить текущую версию Scilab с сайта <http://www.scilab.org>. Развернуть файл в какую либо папку (например, `/usr/lib`), и войдя в папку **Scilab** под `root` выполнить команду `make`. После этого командой `/usr/lib/Scilab 4/bin/scilab` можно запускать программу на выполнение.

1.2. Среда Scilab

После запуска Scilab на экране появиться основное окно приложения. Окно содержит *меню, панель инструментов и рабочую область*. Признаком того, что система готова к выполнению команды, является наличие знака приглашения `-->`, после которого расположен активный (мигающий) курсор. Рабочую область со знаком приглашения обычно называют *командной строкой*. Ввод команд в Scilab осуществляется с клавиатуры. Нажатие клавиши `Enter` заставляет систему выполнить команду и вывести результат (рис. 1.3).

Понятно, что все выполняемые команды не могут одновременно находиться в поле зрения пользователя. Поэтому, просмотреть ту информацию, которая покинула видимую часть окна можно, если воспользоваться стандартными средствами просмотра информации в окне **Windows**, например, полосами прокрутки или клавишами перемещения курсора **Page Up**, **Page Down**.

Клавиши «Стрелку вверх» \uparrow и «Стрелка вниз» \downarrow так же управляют курсором, однако в

Scilab они имеют другое назначение. Эти клавиши позволяют вернуть в командную строку ранее введенные команды или другую входную информацию, так как вся эта информация сохраняется в специальной области памяти. Так, если в пустой активной командной строке нажать клавишу \uparrow , то появится последняя вводимая команда, повторное нажатие вызовет предпоследнюю и так далее. Клавиша \downarrow выводит команды в обратном порядке. Таким образом, можно сказать, что вся информация в рабочей области находится или в зоне просмотра или в зоне редактирования.

Важно знать, что в *зоне просмотра* нельзя ничего исправить или ввести. Единственная допустимая операция это выделение информации с помощью мыши и копирование ее в буфер обмена, например, для дальнейшего помещения в командную строку.

Зона редактирования это фактически командная строка. В ней действуют элементарные приемы редактирования:

- \rightarrow – перемещение курсора вправо на один символ;
- \leftarrow – перемещение курсора влево на один символ;
- **Home** – перемещение курсора в начало строки;
- **End** – перемещение курсора в конец строки;
- **Del** – удаление символа после курсора;
- **Backspace** \leftarrow – удаление символа перед курсором.

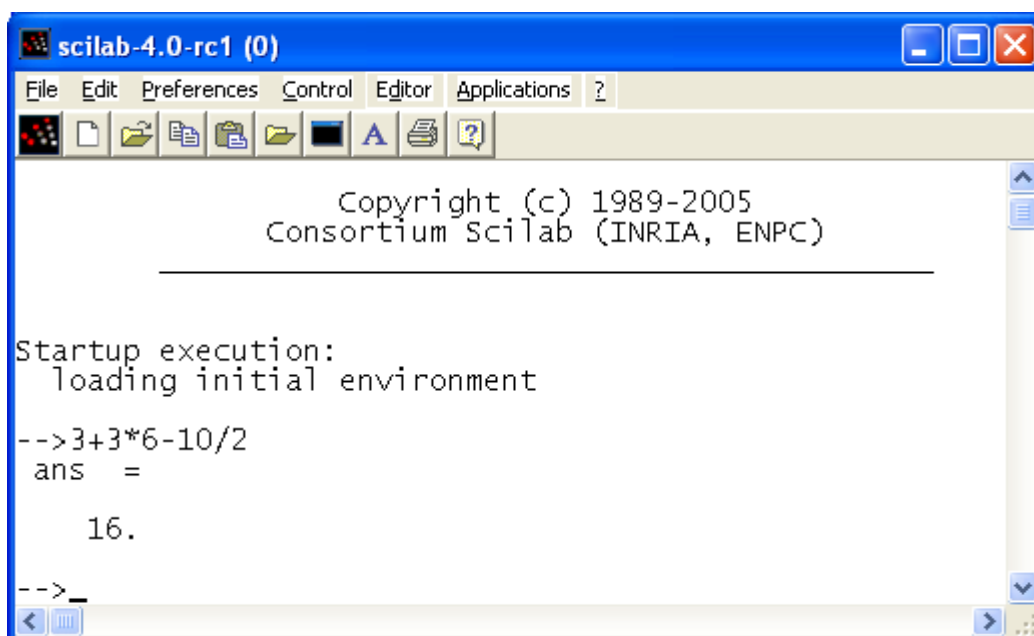


Рис. 1.3 . Выполнение элементарной команды в Scilab

Кроме того, существуют *особенности ввода команд*. Если команда заканчивается точкой с запятой «;», то результат ее действия не отображается в командной строке. В противном случае, при отсутствии знака «;», результат действия команды сразу же выводится в рабочую область (листинг 1.1).

```
-->2.7*3+3.14/2  
ans =  
  
9.67
```

```
-->2.7*3+3.14/2;
```

```
-->
```

Листинг 1.1.

Текущий документ, отражающий работу пользователя с системой **Scilab**, содержащий строки ввода, вывода и сообщения об ошибках принято называть *сессией*. Значения всех переменных, вычисленные в течение текущей сессии, сохраняются в специально зарезервированной области памяти, называемой *рабочим пространством* системы. При желании, определения всех переменных и функций, входящих в текущую сессию, можно сохранить в виде файла, саму сессию сохранить нельзя.

1.3. Основные команды главного меню Scilab

Главное меню системы содержит команды предназначенные для работы с файлами, настройки среды, редактирования команд текущей сессии и получения справочной информации. Кроме того, с помощью главного меню можно создавать, редактировать, выполнять отладку и запускать на выполнение так называемые файлы-сценарии Scilab, а так же работать с графическими приложениям пакета.

1.3.1. Работа с файлами

Пункт меню **File** предназначен для *работы с файлами*. Рассмотрим назначение представленных в нем команд:

- **New Scilab** – открывает новое окно **Scilab**, фактически пакет запускается повторно;
- **Exec...** – запуск на выполнение созданной ранее **Scilab**-программы;
- **Open** – открывает окно для загрузки созданного ранее файла, рисунка или модели;
- **Load** – открывает окно для загрузки файлов, информация в которых хранится в виде машинных кодов, при их открытии в память компьютера загружаются определенные ранее переменные и функции;
- **Save** – сохранение всех определенных в данной сессии переменных и функций в виде файлов;
- **Change Directory** – смена текущего каталога, выводит окно настройки путей файловой системы;

- **Get Change Directory** – имя текущего каталога;
- **Page Setup...** – выводит окно настройки параметров страницы;
- **Print...** – выводит окно настройки параметров печати;
- **Exit** – выход из системы **Scilab**.

1.3.2. Редактирование команд текущей сессии

Пункт меню **Edit** содержит следующие команды:

- **Copy** – копирование выделенного объекта в буфер;
- **Paste** – вставка объекта из буфера;
- **Select All** – выделить все команды сессии;
- **Empty Clipboard** – очистить буфер обмена;
- **History** – группа команд предназначенных для редактирования командной строки.

1.3.3. Настройка среды

Команды настройки среды пакета представлены в меню **Preferences**:

- **Language** – предлагает выбрать из списка язык интерфейса (английский, французский);
- **Colors** – позволяет установить цвет шрифта (**Text**), цвет фона (**Background**) или цвета принятые по умолчанию (**Default System Colors**);
- **Toolbar (F3)** – выводит или удаляет панель инструментов;
- **Files Association** – предлагает установить типы поддерживаемых файлов;
- **Choose Font** – выполняет настройки шрифта (гарнитура, начертание, размер);
- **Clear History** – очищает рабочее пространство;
- **Clear Command Window (F2)** – очищает рабочее окно;
- **Consol (F12)** – активизирует консольное приложение.

1.3.4. Справочная система

Команда главного меню **?** открывает доступ к *справочной системе Scilab*.

В справочной системе информацию можно искать, воспользовавшись содержанием, в списке, упорядоченном по алфавиту, по ключевому слову или фразе.

С помощью команды **Scilab Demos** можно осуществить просмотр демонстрационных примеров.

1.3.5. Редактирование и отладка файлов-сценариев

Файл-сценарий это список команд Scilab сохраненный на диске.

Для подготовки, редактирования и отладки *файлов-сценариев* служит специальный редактор, который можно вызвать, выполнив команду главного меню **Editor**. В результате работы этой команды будет создан *новый файл-сценарий*.

Окно редактора *файлов-сценариев* выглядит как стандартное окно Windows, то есть имеет заголовок, меню, панели инструментов, строку состояния.

Ввод текста в окно редактора *файла-сценария* осуществляется по правилам принятым для команд **Scilab**. Рис. 1.4 содержит пример ввода команд для решения квадратного уравнения $3x^2+5x+4=0$. Не трудно заметить, что точка с запятой «;» ставится после тех команд, которые не требуют вывода значений.

Для *сохранения* введенной информации необходимо выполнить команду **File→Save** из меню редактора. Если информация сохраняется впервые, то появится окно **Save file As....** Ввод имени в поле **File Name** и щелчок по кнопке **Save** приведет к сохранению информации, находящейся в окне редактора. *Файлы-сценариев* сохраняют с расширением **.sce**.

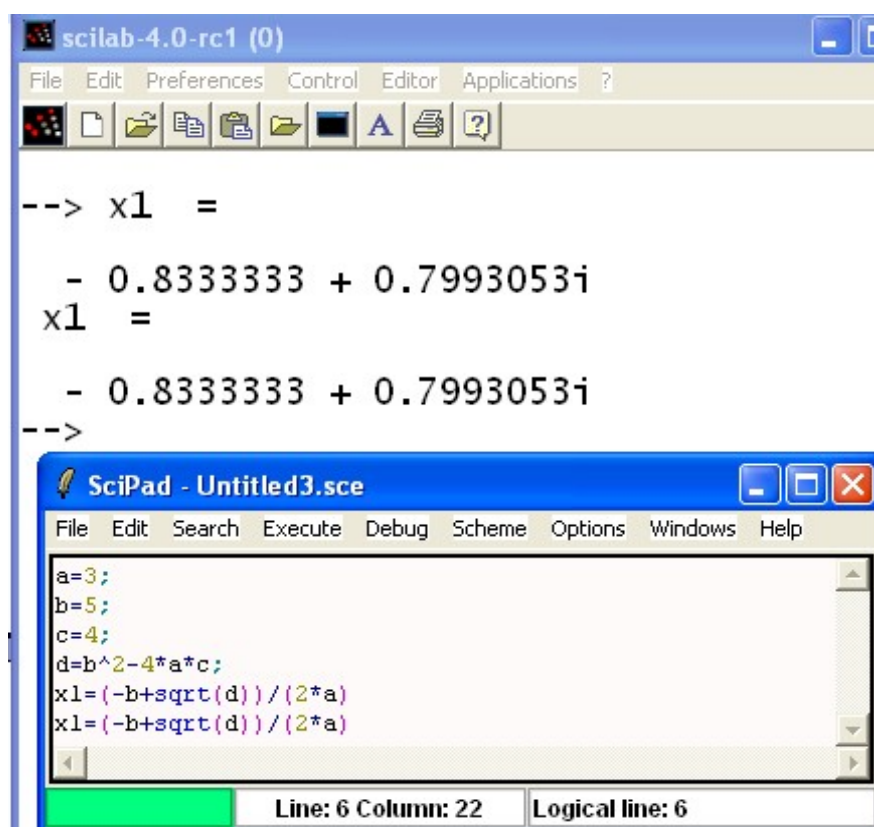


Рис. 1. 4. Выполнение файла-сценария Scilab

Выполнить операторы файла-сценария можно несколькими способами:

- вызвать команду **Execute→Load into Scilab** из меню редактора;
- вызвать команду главного меню **Exec** и указать имя *файла-сценария*.

Все эти действия приведут к появлению в рабочей области результатов вычислений

команд *файла-сценария*.

Отметим, что редактор *файлов-сценариев* имеет возможность работы с множеством окон и обладает принятыми для текстовых редакторов приемами редактирования, на которых подробно мы останавливаться не будем.

Выйти из режима редактирования, можно просто закрыв окно, нажатием крестика в правом верхнем углу.

Открывает ранее созданный файл команда главного меню **File→Open**.

2. Основы работы в Scilab

2.1. Текстовые комментарии

Текстовый комментарий в Scilab это строка, начинающаяся с символов //.

Использовать текстовые комментарии можно как в рабочей области, так и в тексте файла-сценария. Строка после символов // не воспринимается как команда и нажатие клавиши **Enter** приводит к активизации следующей командной строки (листинг 2.1)

```
--> //6+8
```

```
-->
```

Листинг 2.1.

2.2. Элементарные математические выражения

Для выполнения *простейших арифметических операций* в Scilab применяют следующие операторы: + сложение, – вычитание, * умножение, / деление слева направо, \ деление справа налево, ^ возведение в степень.

Вычислить значение арифметического выражения можно, если ввести его в командную строку и нажать клавишу ENTER. В рабочей области появится результат (листинг 2.2).

```
--> 2.35*(1.8-0.25)+1.34^2/3.12
```

```
ans =
```

```
4.2180
```

Листинг 2.2

Если вычисляемое выражение *слишком длинное*, то перед нажатием клавиши ENTER следует набрать три или более точек. Это будет означать продолжение командной строки (листинг 2.3).

```
--> 1+2+3+4+5+6....
```

```
+7+8+9+10+....
```

```
+11+12+13+14+15
```

```
ans =
```

```
120
```

Листинг 2.3

Если символ точки с запятой «;» указан в конце выражения, то результат вычислений не выводится, а активизируется следующая командная строка (листинг 2.4).

```
--> 1+2;
```

```
--> 1+2
```

```
ans =
```

```
3
```

Листинг 2.4

2.3. Переменные в Scilab

В рабочей области Scilab можно определять *переменные*, а затем использовать их в выражениях.

Любая переменная до использования в формулах и выражениях должна быть определена.

Для определения переменной необходимо набрать имя переменной, символ «=» и значение переменной. Здесь знак равенства – это *оператор присваивания*, действие которого не отличается от аналогичных операторов языков программирования. То есть, если в общем виде оператор присваивания записать как

имя переменной = значение выражения

то в *переменную*, имя которой указано слева, будет записано *значение выражения*, указанного справа.

Имя переменной не должно совпадать с именами встроенных процедур, функций и встроенных переменных системы и может содержать до 24 символов. Система различает большие и малые буквы в именах переменных. То есть ABC, abc, Abc, aBc – это имена разных переменных. Выражение в правой части оператора присваивания может быть числом, арифметическим выражением, строкой символов или символьным выражением. Если речь идет о символьной или строковой переменной, то выражение в правой части оператора присваивания следует брать в одинарные кавычки.

Если символ «;» в конце выражения отсутствует, то в качестве результата выводится имя переменной и ее значение. Наличие символа «;» передает управление следующей командной строке. Это позволяет использовать имена переменных для записи промежуточных результатов в память компьютера (листинг 2.5).

```
-->//-----  
-->//Присваивание значений переменным a и b  
--> a=2.3  
a =  
2.3000  
--> b=-34.7  
b =  
-34.7000  
-->//Присваивание значений переменным x и y,  
-->//вычисление значения переменной z  
--> x=1;y=2; z=(x+y)-a/b  
z =  
3.0663  
-->//Сообщение об ошибке – переменная c не определена  
--> c+3/2  
??? Undefined function or variable 'c'.  
-->//-----  
-->//Определение символьной переменной  
--> c='a'  
c =  
a  
-->//Определение строковой переменной  
--> h='мама мыла раму'  
h =  
мама мыла раму
```

Листинг 2.5

Для очистки значения переменной можно применить команду

```
clear имя переменной;
```

Команда `clear`; отменяет определения всех переменных данной сессии. В листинге 2.6

приведены примеры применения этой команды.

```
--> //Определение переменных x и y
--> x=3; y=-1;
--> //Отмена определения переменной x
--> clear x
--> //Переменная x не определена
--> x
??? Undefined function or variable 'x'.
--> //Переменная y определена
--> y
y =
    -1
--> //Определение переменных a и b
--> a=1;b=2;
--> //Отмена определения переменных a и b
--> clear;
--> //Переменные a и b не определены
--> a
!-error 4
undefined variable : a
--> b
!-error 4
undefined variable : b
```

Листинг 2.6

2.4. Системные переменные Scilab

Если команда не содержит знака присваивания, то по умолчанию вычисленное значение присваивается специальной *системной переменной* `ans`. Причем полученное значение можно использовать в последующих вычислениях, но важно помнить, что значение `ans` изменяется после каждого вызова команды без оператора присваивания (листинг 2.7).

```
--> 25.7-3.14
ans =
22.5600
--> //Значение системной переменной равно 22.5600
--> 2*ans
ans =
45.1200
--> //Значение системной переменной увеличено вдвое
--> x=ans^0.3
x =
    3.1355
--> ans
ans =
    45.1200
--> //После использования в выражении значение
--> //системной переменной не изменилось и равно 45.1200
```

Листинг 2.7

Результат последней операции без знака присваивания хранится в переменной `ans`.

Другие *системные переменные* в Scilab начинаются с символа `%`:

- `%i` – мнимая единица ($\sqrt{-1}$);
- `%pi` – число π (3.141592653589793);
- `%e` – число $e=2.7182818$;
- `%inf` – машинный символ бесконечности (∞);
- `%NaN` – неопределенный результат ($0/0$, ∞/∞ , 1^∞ и т.п.);
- `%eps` – условный ноль `%eps=2.220E-16`.

Все перечисленные переменные можно использовать в математических выражениях.

Листинг 2.8 содержит пример вычисления выражения $F=\cos(\pi/3)+(a-b)\cdot e^2$.

```
-->a=5.4;b=0.1;
-->F=cos(%pi/3)+(a-b)*%e^2
F =
    39.661997
```

Листинг 2.8

В листинге 2.9 показан пример неверного обращения к системной переменной.

```
-->sin(pi/2)
!--error 4
undefined variable : pi
```

Листинг 2.9

2.5. Ввод вещественного числа и представление результатов вычислений

Числовые результаты могут быть представлены с *плавающей* (например, $-3.2E-6$, $-6.42E+2$), или с *фиксированной* (например, 4.12, 6.05, -17.5489) точкой. Числа в формате с плавающей точкой представлены в экспоненциальной форме $mE\pm p$, где m – мантисса (целое или дробное число с десятичной точкой), p – порядок (целое число). Для того, чтобы перевести число в экспоненциальной форме к обычному представлению с фиксированной точкой, необходимо **мантиссу умножить на десять в степени порядок**.

Например,

$$\begin{aligned} -6.42E+2 &= -6.42 \cdot 10^2 = -642 \\ 3.2E-6 &= 3.2 \cdot 10^{-6} = 0.0000032 \end{aligned}$$

При вводе вещественных чисел для отделения дробной части используется точка.

Примеры ввода и вывода вещественных чисел показаны в листинге 2.10.

```
-->0.123
ans =
    0.123
-->-6.42e+2
```

```
ans =  
- 642.  
-->3.2e-6  
ans =  
0.0000032
```

Листинг 2.10.

Листинг 2.11 содержит пример вывода значения системной переменной π и некоторой переменной q . Не трудно заметить, что Scilab в качестве результата выводит только восемь значащих цифр.

```
-->%pi  
%pi =  
3.1415927  
-->q=0123.4567890123456  
q =  
123.45679
```

Листинг 2.11

Восемь значащих цифр – это *формат вывода вещественного числа по умолчанию*. Для того, чтобы контролировать количество выводимых на печать разрядов применяют команду `printf` с заданным форматом, который соответствует правилам принятым для этой команды в языке C. Листинг 2.12 содержит несколько примеров вызова команды `printf`.

```
-->printf("%1.12f", %pi)  
3.141592653590  
-->printf("%1.15f", %pi)  
3.141592653589793  
-->printf("%1.2f", q)  
123.46  
-->printf("%1.10f", q)  
123.4567890123  
-->//По умолчанию 6 знаков после запятой  
-->printf("%f", q)  
123.456789
```

Листинг 2.12

2.6. Функции в Scilab

Все функции, используемые в Scilab, можно разделить на два класса:

- *встроенные;*
- *определенные пользователем.*

В общем виде *обращение к функции* в Scilab имеет вид:

```
имя_переменной = имя_функции(переменная1 [, переменная2, ...])
```

где

- `имя_переменной` – переменная, в которую будут записаны результаты работы функции; этот параметр может отсутствовать, тогда значение, вычисленное функцией будет присвоено системной переменной `ans` ;

- имя_функции – имя встроенной функции или ранее созданной пользователем;
- переменная1, переменная2, ... – список аргументов функции.

2.6.1. Элементарные математические функции

Пакет Scilab снабжен достаточным количеством всевозможных *встроенных функций*, знакомство с которыми будет происходить в следующих разделах. Здесь приведем, только элементарные математические функции, используемые чаще всего (табл. 2.1).

Табл. 2.1. Элементарные математические функции MATLAB

Функция	Описание функции
Тригонометрические	
sin(x)	синус числа x
cos(x)	косинус числа x
tan(x)	тангенс числа x
cotg(x)	котангенс числа x
asin(x)	арксинус числа x
acos(x)	арккосинус числа x
atan(x)	арктангенс числа x
Экспоненциальные	
exp(x)	Экспонента числа x
log(x)	Натуральный логарифм числа x
Другие	
sqrt(x)	корень квадратный из числа x
abs(x)	модуль числа x
log10(x)	десятичный логарифм от числа x
log2(x)	логарифм по основанию два от числа x

Листинг 2.13 содержит пример вычисления выражения $z = \sqrt{\left| \sin\left(\frac{x}{y}\right) \right|} \cdot e^{x^y}$.

```
-->x=1.2;y=0.3;

-->z=sqrt(abs(sin(x/y)))*exp(x^y)
z =

    2.5015073
```

Листинг 2.13.

2.6.2. Функции, определенные пользователем

В первой главе мы уже упоминали о *файлах-сценариях* и даже создавали небольшую программу, которая решала конкретное квадратное уравнение. Но в эту программу невозможно было передать входные параметры, то есть это был обычный список команд, воспринимаемый системой как единый оператор.

Функция, как правило, предназначена для неоднократного использования, она имеет *входные параметры* и не выполняется без их предварительного задания. Рассмотрим несколько способов создания функций в Scilab.

Первый способ это применение оператора

```
def f (' [имя1, ..., имяN] = имя_функции (переменная_1, ..., переменная_M) ',
      ' имя = выражение; ...; имя1 = выражение1; ...; имяN = выражениеN ' )
```

где $\text{имя1}, \dots, \text{имяN}$ – список выходных параметров (от 1 до N), то есть переменных, которым будет присвоен конечный результат вычислений, имя_функции – имя с которым эта функция будет вызываться, $\text{переменная}_1, \dots, \text{переменная}_M$ – входные параметры (от 1 до M).

Так, в листинге 2.14 приведен самый простой способ применения оператора `def f`. Здесь

показано как создать и применить функцию для вычисления выражения $z = \sqrt{\left| \sin\left(\frac{x}{y}\right) \right|} \cdot e^{x \cdot y}$

(значение этого выражения уже было вычислено в листинге 2.13).

```
-->def f (' z=fun1 (x, y) ', ' z=sqrt (abs (sin (x/y) )) *exp (x^y) ');
-->x=1.2; y=0.3; z=fun1 (x, y)
z
=
2.5015073
```

Листинг 2.14

Листинг 2.15 содержит пример создания и применения функции, вычисляющей площадь треугольника со сторонами a , b и c по формуле Герона $S = \sqrt{(p-a) \cdot (p-b) \cdot (p-c)}$, где

$$p = \frac{(a+b+c)}{2} .$$

```
-->def f (' S=G (a, b, c) ', ' p=(a+b+c) /2; S=sqrt ((p-a) * (p-b) * (p-c) ) ');
-->G (2, 3, 3)
ans
=
1.4142136
```

Листинг 2.15

С помощью функции созданной в листинге 2.16 можно найти корни квадратного уравнения.

```
-->def f (' [x1, x2]=korni (a, b, c) ',
      ' d=b^2-4*a*c; x1=(-b+sqrt (d) ) /2/a; x2=(-b-sqrt (d) ) /2/a ');
-->[x1, x2]=korni (-2, -3, 5)
x2
=
1.
x1
=
2.5
```

Листинг 2.16

Второй способ создания функции это применение конструкции вида:

```
function [имя1, ..., имяN] = имя_функции (переменная_1, ..., переменная_M)
    тело функции
endfunction
```

где $\text{имя1}, \dots, \text{имяN}$ – список выходных параметров (от 1 до N), то есть переменных, которым будет присвоен конечный результат вычислений, имя_функции – имя с которым эта функция будет вызываться, $\text{переменная}_1, \dots, \text{переменная}_M$ – входные параметры (от 1 до M).

Все имена переменных внутри функции, а так же имена из списка входных и выходных параметров воспринимаются системой как *локальные*, то есть эти переменные считаются определенными только внутри функции.

Вообще говоря, функции в Scilab играют роль *подпрограмм*. Поэтому целесообразно набирать их тексты в редакторе и сохранять в виде отдельных файлов. Причем имя файла должно обязательно совпадать с именем функции. Расширение файлам-функциям обычно присваивают *.sci или *.sce.

Обращение к функции осуществляется так же, как и к любой другой встроенной функции системы, то есть из командной строки. Однако функции, хранящиеся в отдельных файлах должны быть *предварительно загружены* в систему, например при помощи оператора `exec` (и имя_файла) или командой главного меню **File\Exec...**, что в общем, одно и то же.

ЗАДАЧА 2.1. Решить кубическое уравнение.

Кубическое уравнение

$$ax^3 + bx^2 + cx + d = 0 \quad (2.1)$$

после деления на a принимает канонический вид:

$$x^3 + rx^2 + sx + t = 0 \quad , \quad (2.2)$$

где

$$r = \frac{b}{a} \quad , \quad s = \frac{c}{a} \quad , \quad t = \frac{d}{a} \quad .$$

В уравнении (2.2) сделаем замену

$$x = y - \frac{r}{3}$$

и получим следующее приведенное уравнение:

$$y^3 + py + q = 0 \quad , \quad (2.3)$$

где

$$p = \frac{3s - r^2}{3} \quad , \quad q = \frac{2r^3}{27} - \frac{rs}{3} + t \quad .$$

Число действительных корней приведенного уравнения (2.3) зависит от знака дискриминанта $D = \left(\frac{p}{3}\right)^3 + \left(\frac{q}{2}\right)^3$ (табл. 2.3).

Табл. 2.2. Количество корней кубического уравнения

Дискриминант	Количество действительных корней	Количество комплексных корней
$D \geq 0$	1	2
$D < 0$	3	-

Корни приведенного уравнения могут быть рассчитаны по формулам Кардано:

$$y_1 = u + v, y_2 = \frac{-(u+v)}{2} + \frac{(u-v)}{2} i \sqrt{3}, y_3 = \frac{-(u+v)}{2} - \frac{(u-v)}{2} i \sqrt{3} \quad (2.4)$$

Здесь

$$u = \sqrt[3]{\frac{-q}{2} + \sqrt{(D)}}, v = \sqrt[3]{\frac{-q}{2} - \sqrt{(D)}}$$

Список команд, реализующий описанный выше способ решения кубического уравнения, представлен в виде функции на листинге 2.17. Листинг 2.18 содержит фрагмент рабочей области, демонстрирующий вызов функции и вывод результатов ее работы.

```
function [x1, x2, x3]=cub(a, b, c, d)
r=b/a;
s=c/a;
t=d/a;
p=(3*s-r^2)/3;
q=2*r^3/27-r*s/3+t;
D=(p/3)^3+(q/2)^2;
u=(-q/2+sqrt(D))^(1/3);
v=(-q/2-sqrt(D))^(1/3);
y1=u+v;
y2=-(u+v)/2+(u-v)/2*i*sqrt(3);
y3=-(u+v)/2-(u-v)/2*i*sqrt(3);
x1=y1-r/3;
x2=y2-r/3;
x3=y3-r/3;
endfunction
```

Листинг 2.17

```
-->exec('C:\Ksu\Scilab\scilab-4.0-rc1\cub.sce');
-->disp('exec done');
Warning :redefining function: cub
exec done
-->[x1, x2, x3]=cub(3, -20, -3, 4)
x3 =
    0.3880206
x2 =
   - 0.5064407
x1 =
    6.7850868
```

Листинг 2.18

3. Массивы и матрицы в Scilab. Решение задач линейной алгебры

Для работы с множеством данных удобно использовать массивы. Например, можно создать массив для хранения числовых или символьных данных. В этом случае, вместо создания переменной, для хранения каждого данного, достаточно создать один массив, где каждому элементу будет присвоен порядковый номер.

Таким образом, *массив* – множественный тип данных, состоящий из фиксированного числа элементов. Как и любой другой переменной, массиву должно быть присвоено *имя*.

Переменную, представляющую собой просто список данных, называют *одномерным* массивом или *вектором*. Для доступа к данным, хранящимся в определенном элементе массива, необходимо указать имя массива и порядковый номер этого элемента, называемый *индексом*.

Если возникает необходимость хранения данных в виде таблиц, в формате строк и столбцов, то необходимо использовать *двумерные* массивы (*матрицы*). Для доступа к данным, хранящимся в таком массиве, необходимо указать имя массива и *два индекса*, первый должен соответствовать номеру строки, а второй номеру столбца в которых хранится необходимый элемент.

Значение нижней границы индексации в Scilab равно единице. Индексы могут быть только целыми положительными числами.

3.1. Ввод и формирование массивов и матриц

Самый простой способ *задать одномерный массив* в Scilab имеет вид

$$[\text{name}] = X_n : dX : X_k^1$$

где *name* – имя переменной, в которую будет записан сформированный массив, X_n – значение первого элемента массива, X_k – значение последнего элемента массива, dX – шаг, с помощью которого формируется каждый следующий элемент массива, то есть значение второго элемента составит $X_n + dX$, третьего $X_n + dX + dX$ и так далее до X_k .

Возможен и такой способ *определения одномерного массива*:

$$[\text{name}] = X_n : X_k$$

Если параметр dX в конструкции отсутствует, это означает, что по умолчанию он принимает значение равное единице, то есть каждый следующий элемент массива равен значению предыдущего плюс один.

Примеры определения одномерных массивов таким способом приведены в листинге 3.1.

Переменную заданную как массив можно использовать в арифметических выражениях и в качестве аргумента математических функций. Результатом работы таких операторов

¹ Здесь и далее квадратные скобки (если не сказано иначе) означают, что выражение, указанное в них может отсутствовать.

являются массивы (листинг 3.1).

```
--> Xn=-3.5;dX=1.5;Xk=4.5;
--> X=Xn:dX:Xk
X =
-3.5000 -2.0000 -0.5000 1.0000 2.5000 4.0000
--> Y=sin(X/2)
Y =
-0.9840 -0.8415 -0.2474 0.4794 0.9490 0.9093
--> A=0:5
A =
0 1 2 3 4 5
--> 0:5
ans =
0 1 2 3 4 5
--> ans/2+pi
ans =
3.1416 3.6416 4.1416 4.6416 5.1416 5.6416
```

Листинг 3.1

Еще один способ задания векторов и матриц в Scilab это их *поэлементный ввода*.

Так для *определения вектор–строки* следует ввести имя массива, а затем после знака присваивания, в квадратных скобках через пробел или запятую перечислить элементы массива:

$$[\text{name}] = x_1 \ x_2 \ \dots \ x_n \text{ или } [\text{name}] = x_1, \ x_2, \ \dots, \ x_n$$

Пример ввода вектора–строки приведен в листинге 3.2.

```
--> V=[1 2 3 4 5]
V =
1 2 3 4 5
--> W=[1.1,2.3,-0.1,5.88]
W =
1.1000 2.3000 -0.1000 5.8800
```

Листинг 3.2

Элементы *вектора–столбца* вводятся через точку с запятой: $[\text{name}] = x_1; \ x_2; \ \dots; \ x_n$

Листинг 3.3 содержит пример ввода вектора–столбца.

```
--> X=[1;2;3]
X =
1
2
3
```

Листинг 3.3

Обратиться к элементу вектора можно, указав имя массива и порядковый номер элемента в круглых скобках: `name(индекс)`. Листинг 3.4 содержит пример обращения к элементам массива.

```
--> W=[1.1,2.3,-0.1,5.88];
--> W(1)+2*W(3)
ans =
0.9000
```

Листинг 3.4

Ввод элементов *матрицы* так же осуществляется в квадратных скобках, при этом элементы строки отделяются друг от друга пробелом или запятой, а строки разделяются между собой точкой с запятой:

$$[\text{name}] = [x_{11}, x_{12}, \dots, x_{1n}; x_{21}, x_{22}, \dots, x_{2n}; \dots; x_{m1}, x_{m2}, \dots, x_{mn};$$

Обратиться к *элементу матрицы* можно, указав после имени матрицы, в круглых скобках, через запятую, номер строки и номер столбца на пересечении которых элемент расположен: `name(индекс1, индекс2)`.

Примеры задания матриц и обращение к их элементам в листинге 3.5.

```
--> A=[1 2 3;4 5 6;7 8 9]
A =
     1     2     3
     4     5     6
     7     8     9
--> A(1,2)^A(2,2)/A(3,3)
ans =
     3.5556
```

Листинг 3.5

Кроме того, матрицы и векторы можно *формировать*, составляя их из ранее заданных матриц и векторов (листинг 3.6).

```
--> v1=[1 2 3]; v2=[4 5 6]; v3=[7 8 9];
--> //Горизонтальная конкатенация векторов-строк.
--> V=[v1 v2 v3]
V =
     1     2     3     4     5     6     7     8     9
--> //Вертикальная конкатенация векторов-строк, результат матрица
--> V=[v1; v2; v3]
V =
     1     2     3
     4     5     6
     7     8     9
--> //Горизонтальная конкатенация матриц
--> M=[V V V]
M =
     1     2     3     1     2     3     1     2     3
     4     5     6     4     5     6     4     5     6
     7     8     9     7     8     9     7     8     9
--> //Вертикальная конкатенация матриц
--> M=[V;V]
M =
     1     2     3
     4     5     6
     7     8     9
     1     2     3
     4     5     6
     7     8     9
```

Листинг 3.6

Важную роль при работе с матрицами играет знак двоеточия «:». Указывая его вместо индекса при обращении к массиву, можно иметь доступ к группам его элементов (листинг

3.7).

```

--> A=[5 7 6 5; 7 10 8 7;6 8 10 9;5 7 9 10]
A = //Пусть задана матрица A
5 7 6 5
7 10 8 7
6 8 10 9
5 7 9 10
--> //Выделить из матрицы A второй столбец
--> A(:,2)
ans =
7
10
8
7
--> //Выделить из матрицы A третью строку
--> A(3,:)
ans =
6 8 10 9
--> //Выделить из матрицы A подматрицу M
--> M=A(3:4,2:3)
M =
8 10
7 9
--> //Удалить из матрицы A второй столбец
--> A(:,2)=[]
A =
5 8 10
7 7 9
6 10 9
5 9 10
--> //Удалить из матрицы A третью строку
--> A(3,:)=[]
A =
5 8 10
7 7 9
5 9 10
--> //Представить матрицу M в виде вектора-столбца
--> v=M(:)
v =
8
7
10
9
--> //Выделить из вектора v элементы со второго по четвертый
--> b=v(2:4)
b =
7
10
9
--> //Удалить из массива b второй элемент
--> b(2)=[];

```

Листинг 3.7

3.2. Действия над матрицами

Для работы с матрицами и векторами в Scilab предусмотрены следующие операции:

- + – сложение;
- – – вычитание²;
- ' – транспонирование;
- * – матричное умножение³;
- * – умножение на число;
- ^ – возведение в степень⁴;
- \ – левое деление, $(A \setminus B) \Rightarrow (A^{-1} \cdot B)$, операция может быть применима для решения уравнения матричного уравнения вида $A \cdot X = B$;
- / – правое деление, $(B / A) \Rightarrow (B \cdot A^{-1})$, используют для решения матричных уравнений вида $X \cdot A = B$;
- .* – поэлементное умножение матриц;
- .^ – поэлементное возведение в степень;
- .\ – поэлементное левое деление;
- ./ – поэлементное правое деление;

Листинг 3.8. содержит пример действий над матрицами.

```
-->A=[1 2 0;-1 3 1;4 -2 5];
-->B=[-1 0 1;2 1 1;3 -1 -1];
-->//Вычислить (A^T+B)^2 - 2A(0.5B^T-A)
-->(A'+B)^2-2*A*(1/2*B'-A)
ans =
    10.    8.    24.
    11.   20.   35.
    63.  -30.   68.
```

Листинг 3.8

Кроме того, если к некоторому заданному вектору или матрице применить *математическую функцию*, то результатом будет новый вектор или матрица той же размерности, но элементы будут преобразованы в соответствии с заданной функцией (листинг 3.9).

² Эти операции определены для матриц одной размерности или векторов одного типа, то есть суммировать (вычитать) можно либо векторы–столбцы, либо векторы–строки одинаковой длины.

³ Операция умножения вектора на вектор определена только для векторов одинакового размера, причем один из них должен быть вектором–столбцом, а второй вектором–строкой. Матричное умножение допустимо, если количество строк в одной матрице совпадает с количеством столбцов в другой.

⁴ Возвести матрицу в n-ю степень, значит умножить ее саму на себя n раз. При этом целочисленный показатель степени может быть как положительным, так и отрицательным. В первом случае выполняется алгоритм умножения матрицы на себя указанное число раз, во втором умножается на себя матрица *обратная к данной*.

```
--> x=[0.1 -2.2 3.14 0 -1];
--> sin(x)
ans =
0.0998 -0.8085 0.0016 0 -0.8415
```

Листинг 3.9

3.3. Специальные матричные функции

Для работы с матрицами и векторами в Scilab существуют специальные функции.

Рассмотрим наиболее часто используемые из них:

- `length(X)` – определяет количество элементов матрицы X , если X – вектор, его длину (листинг 3.10);

```
--> V=[-1 0 3 -2 1 -1 1]; //Вектор-строка
--> length(V) //Длина вектора
ans =
7
```

Листинг 3.10

- `prod(X)` – вычисляет произведение элементов матрицы или вектора X (листинг 3.11);

```
--> V=[1, 2, 3];
--> prod(V) %Произведение элементов вектора
ans =
6
```

Листинг 3.11

- `sum(X)` – вычисляет сумму элементов матрицы или вектора X , кроме того с помощью этой функции можно вычислить скалярное произведение векторов (листинг 3.12);

```
--> V=[-1 0 3 -2 1 -1 1];
--> sum(V) //Сумма элементов вектора
ans =
1
--> //Частный случай. Вычисление скалярного произведения
--> a=[1 2 3]; b=[2 0 1];
--> sum(a.*b)
ans =
5.
```

Листинг 3.12

- `min(X)` – находит минимальный элемент матрицы или вектора X (листинг 3.13);
- `max(X)` – находит максимальный элемент матрицы или вектора X (листинг 3.13);

```
--> V=[-1 0 3 -2 1 -1 1];
--> min(V) //Минимальный элемент
ans =
-2
--> max(V) //Максимальный элемент
ans =
3
```

Листинг 3.13

- `mean(X)` – определяет среднее арифметическое матрицы или вектора X (листинг 3.14);

```
--> V=[-1 0 3 -2 1 -1 1];
--> mean(V)%Среднее значение массива V
ans =
    0.1429
--> sum(V)/length(V)%То же что и mean(V)
ans =
    0.1429
```

Листинг 3.14

- `sort(X)` – выполняет упорядочивание массива X , если X – матрица, сортировка выполняется по столбцам (листинг 3.15);

```
-->b
b =
    2.    0.    1.
-->sort(b) //Сортировка по убыванию
ans =
    2.    1.    0.
-->-sort(-b) //Сортировка по возрастанию
ans =
    0.    1.    2.
-->A
A =
    1.    2.    0.
   -1.    3.    1.
    4.   -2.    5.
-->sort(A) //Сортировка матрицы
ans =
    5.    2.    0.
    4.    1.   -1.
    3.    1.   -2.
```

Листинг 3.15

- `eye(n, m)` – возвращает единичную матрицу соответствующей размерности (листинг 3.16);

```
-->eye(3,3)
ans =
    1.    0.    0.
    0.    1.    0.
    0.    0.    1.-->eye(5,1)
ans =
    1.
    0.
    0.
    0.
    0.
```

Листинг 3.16

- `ones(n, m)` – формирует матрицу, состоящую из единиц (листинг 3.17);

```
-->m=3; n=2;
```

```
-->X=ones(m,n)
X =
    1.    1.
    1.    1.
    1.    1.
```

Листинг 3.17

- `zeros(n, m)` – возвращает нулевую матрицу соответствующей размерности (листинг 3.18);

```
-->zeros(3,2)
ans =
    0.    0.
    0.    0.
    0.    0.
```

Листинг 3.18

- `diag(V [, k])` – возвращает квадратную матрицу с элементами V на главной диагонали или на k -й; функция `diag(A [, k])`, где A ранее определенная матрица, в качестве результата выдаст вектор столбец, содержащий элементы главной или k -ой диагонали матрицы A (листинг 3.19);

```
--> V=[1,2,3];
--> diag(V)//Диагональная матрица, V на главной диагонали
ans =
1  0  0
0  2  0
0  0  3
-->//Диагональная матрица, V на первой диагонали (выше главной)
--> diag(V,1)
ans =
0  1  0  0
0  0  2  0
0  0  0  3
0  0  0  0
-->//Диагональная матрица, V на первой диагонали (ниже главной)
--> diag(V,-1)
ans =
0  0  0  0
1  0  0  0
0  2  0  0
0  0  3  0
--> A=[-1 2 0 ;2 1 -1 ;2 1 3]
A =
-1 2  0
 2 1 -1
 2 1  3
--> diag(A) //Главная диагональ матрицы A
ans =
-1
 1
 3
```

Листинг 3.19

- `rand([n, m, p, ...])` – возвращает матрицу случайных чисел, `rand` без аргументов возвращает одно случайное число (листинг 3.20);

```
--> rand(3,3)//Квадратная матрица случайных чисел
ans =
0.9501  0.4860  0.4565
0.2311  0.8913  0.0185
0.6068  0.7621  0.8214
--> R=rand(2,2,2)//Многомерный массив случайных чисел
R(:,:,1) =
0.9355  0.4103
0.9169  0.8936
R(:,:,2) =
0.0579  0.8132
0.3529  0.0099
--> rand //Одно случайное число
ans =
0.4057
```

Листинг 3.20

- `cat(n, A, B, [C, ...])` – объединяет матрицы A и B или все входящие матрицы A, B, C, ... При `n=1` по строкам, при `n=2` по столбцам. То же что `[A; B]` или `[A, B]` (листинг 3.21);

```
--> A=[1 2;3 4];
--> B=[5 6 ;7 8];
--> cat(2,A,B)%Объединение матриц
ans =
1  2  5  6
3  4  7  8
--> cat(1,A,B) %Объединение матриц
ans =
1  2
3  4
5  6
7  8
```

Листинг 3.21

- `tril(A [, k])` – формирует из матрицы A нижнюю треугольную матрицу начиная с главной или с `k`-й диагонали (листинг 3.22);

```
--> A=[1 2 3;4 5 6;7 8 9]
A =
1  2  3
4  5  6
7  8  9
--> tril(A)//Нижняя треугольная матрица, начиная с главной
диагонали
ans =
1  0  0
4  5  0
7  8  9
--> tril(A,0)//Тоже что и tril(A)
ans =
```

```

1  0  0
4  5  0
7  8  9
--> tril(A,1)//Нижняя треугольная матрица,
--> //начиная с первой диагонали (выше главной)
ans =
1  2  0
4  5  6
7  8  9
--> tril(A,-2) //Нижняя треугольная матрица,
--> //начиная со второй диагонали (ниже главной)
ans =
0  0  0
0  0  0
7  0  0

```

Листинг 3.22

- `triu(A [, k])` – формирует из матрицы A верхнюю треугольную матрицу начиная с главной или с k-й диагонали (листинг 3.23);

```

--> A=[1 2 3;4 5 6;7 8 9];
--> triu(A)//Верхняя треугольная матрица
ans =
1  2  3
0  5  6
0  0  9
--> triu(A,2) //Верхняя треугольная матрица,
--> //начиная со второй диагонали (выше главной)
ans =
0  0  3
0  0  0
0  0  0
--> triu(A,-1) //Верхняя треугольная матрица,
--> //начиная с первой диагонали (ниже главной)
ans =
1  2  3
4  5  6
0  8  9

```

Листинг 3.23

- `size(A)` – определяет число строк и столбцов матрицы A, результатом ее работы является вектор [n, m] (листинг 3.24);

```

--> A=[2 4;1 3;7 9;5 8];
--> size(A)//Размерность матрицы A
ans =
4  2
--> V=[2 4 6 8 1 3 5 7];
--> size(V)//Размерность вектора-строки V
ans =
1  8
--> size(V')//Размерность вектора-столбца V'
ans =
8  1
--> //Формирование нулевой матрицы, такого же размера как и A

```

```
--> zeros(size(A))
ans =
0 0
0 0
0 0
0 0
-->//Формирование единичного вектора, такого же размера как и V
--> ones(size(V))
ans =
1 1 1 1 1 1 1 1
```

Листинг 3.24

- `det(A)` – вычисляет определитель квадратной матрицы A (листинг 3.25);

```
--> A=[3 2;4 3];
--> det(A)//Определитель матрицы
ans =
1
```

Листинг 3.25

- `trace(A)` – вычисляет след матрицы A , то есть сумму элементов главной диагонали (листинг 3.26);

```
--> A=[1 2 3;4 -2 1;7 0 -1]
A =
1 2 3
4 -2 1
7 0 -1
--> trace(A)//След матрицы A
ans =
-2
-->//Сумма элементов главной диагонали, то же что и trace(A)
--> sum(diag(A))
ans =
-2
```

Листинг 3.26

- `inv(A)` – возвращает матрицу обратную к A (листинг 3.27);

```
--> A=[2 1 -5 1;1 -3 0 -6;0 2 -1 2;1 4 -7 6];
--> P=inv(A)//Матрица обратная к A
P =
1.3333 -0.6667 0.3333 -1.0000
-0.0741 0.2593 1.1481 -0.1111
0.3704 -0.2963 0.2593 -0.4444
0.2593 -0.4074 -0.5185 -0.1111
--> A*P //Проверка A*P=E
ans =
1.0000 -0.0000 -0.0000 0.0000
0 1.0000 0.0000 0.0000
0.0000 -0.0000 1.0000 -0.0000
0.0000 -0.0000 -0.0000 1.0000
```

Листинг 3.27

- `linsolve(A, b)` – возвращает решение системы линейных уравнений $Ax=b$ (листинг 3.28);

```

--> A=[1 2 3;-2 -4 -6]; b=[5;6];
--> //Система не имеет решений
-->linsolve(A,b)
WARNING:Conflicting linear constraints!
ans =
    []
-->//-----
--> A=[2 -1 1;3 2 -5;1 3 -2]; b=[0;1;4];
--> x=linsolve(A,b)//Решение линейной системы
x =
0.4643
1.6786
0.7500
--> A*x //Решение верно
ans =
0.0000
1.0000
4.0000

```

Листинг 3.28

- `rref(A)` – осуществляет приведение матрицы A к треугольной форме, используя метод исключения Гаусса (листинг 3.29);

```

--> A=[3 -2 1 5;6 -4 2 7;9 -6 3 12]
A =
3  -2  1  5
6  -4  2  7
9  -6  3  12
--> rref(A)
ans =
1.0000    -0.6667    0.3333    0
0          0          0          1.0000
0          0          0          0

```

Листинг 3.29

3.4. Решение некоторых задач алгебры матриц

Матричное уравнение это уравнение вида $A \cdot X = B$ или $X \cdot A = B$, где X это неизвестная матрица. Если умножить матричное уравнение на матрицу обратную к A , то оно примет вид: $A^{-1} A X = A^{-1} B$ или $X A A^{-1} = B A^{-1}$. Так как $A^{-1} A = A A^{-1} = E$, а $E \cdot X = X \cdot E = X$, то неизвестную матрицу X можно вычислить так: $X = A^{-1} B$ или $X = B A^{-1}$. Понятно, что матричное уравнение имеет единственное решение если A и B – квадратные матрицы n -го порядка и определитель матрицы A не равен нулю.

ЗАДАЧА 3.1. Решить матричные уравнения $A \cdot X = B$ и $X \cdot A = B$ выполнить проверку.

Как решить матричное уравнение в Scilab, показано в листинге 3.30.

```

-->A=[3 2;4 3]
A =
3.  2.
4.  3.
-->B=[-1 7;3 5]
B =

```

```

- 1.      7.
  3.      5.
-->// Решение матричного уравнения AX=V
-->// Первый способ
-->X=A\V
X =
- 9.      11.
 13.     -13.
-->// Второй способ
-->X=inv(A)*V
X =
- 9.      11.
 13.     -13.
-->// Решение матричного уравнения XA=V
-->// Первый способ
-->X=V/A
X =
- 31.     23.
- 11.      9.
-->// Первый способ
-->X=B*inv(A)
X =
- 31.     23.
- 11.      9.
-->// Второй способ
-->X*A-B
ans =
  0.      0.
  0.      0.

```

Листинг 3.30**3.5. Решение систем линейных алгебраических уравнений**

Система m уравнений с n неизвестными вида

$$a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1,$$

$$a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2,$$

.....

$$a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n = b_m,$$

называется *системой линейных уравнений*, причем x_j – неизвестные, a_{ij} – коэффициенты при неизвестных, b_i – свободные коэффициенты ($i=1..m$, $j=1..n$). Система из m линейных уравнений с n неизвестными может быть описана при помощи матриц: $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$, где $\mathbf{x} = \{x_j\}$ – вектор неизвестных, $\mathbf{A} = \{a_{ij}\}$ – матрица коэффициентов при неизвестных или матрица системы, $\mathbf{b} = \{b_i\}$ – вектор свободных членов системы или вектор правых частей ($i=1..m$, $j=1..n$). Совокупность всех решений системы (x_1, x_2, \dots, x_n) , называется *множеством решений* или просто *решением системы*.

ЗАДАЧА 3.2. Решить СЛАУ при помощи правила Крамера:

$$\begin{aligned} 2x_1 + x_2 - 5x_3 + x_4 &= 8, \\ x_1 - 3x_2 - 6x_4 &= 9, \\ 2x_2 - x_3 + 2x_4 &= -5, \\ x_1 + 4x_2 - 7x_3 + 6x_4 &= 0. \end{aligned}$$

Правило Крамера заключается в следующем. Если определитель $\Delta = \det A$ матрицы системы из n уравнений с n неизвестными $A \cdot x = b$ отличен от нуля, то система имеет единственное решение x_1, x_2, \dots, x_n , Определяемое по формулам Крамера $x_i = \Delta_i / \Delta$, где Δ_i – определитель матрицы, полученной из матрицы системы A заменой i -го столбца столбцом свободных членов b . Текст файла–сценария с решением задачи по формулам Крамера приведен в листинге 3.31.

```
A=[2 1 -5 1;1 -3 0 -6;0 2 -1 2;1 4 -7 6]; //Матрица коэффициентов
b=[8;9;-5;0]; //Вектор свободных коэффициентов
//Первая вспомогательная матрица
A1=A;A1(:,1)=b;
//Вторая вспомогательная матрица
A2=A;A2(:,2)=b;
//Третья вспомогательная матрица
A3=A;A3(:,3)=b;
//Четвертая вспомогательная матрица
A4=A;A4(:,4)=b;
//Главный определитель отличен от нуля
D=det(A);
//Определители вспомогательных матриц
d(1)=det(A1);
d(2)=det(A2);
d(3)=det(A3);
d(4)=det(A4);
//Вектор неизвестных
x=d/D
//Проверка
P=A*x-b
```

Листинг 3.31

Результаты работы S-файла показаны в листинге 3.31.

```
-->exec('C:\Program Files\scilab-4.0-rc1\bin\kramer.sce');
-->disp('exec done');
x =
  3.
 - 4.
 - 1.
  1.
P =
  1.0D-14 *
  0.1776357
  0.
 - 0.0888178
  0.1554312
exec done
```

Листинг 3.32

ЗАДАЧА 3.3. Решить СЛАУ из задачи 1 методом обратной матрицы.

Метод обратной матрицы: для системы из n линейных уравнений с n неизвестными $A \cdot x = b$, при условии, что определитель матрицы A не равен нулю, единственное решение можно представить в виде $x = A^{-1} \cdot b$.

Текст файла–сценария в листинге 3.33, результаты его работы в листинге 3.34.

```
A=[2 1 -5 1;1 -3 0 -6;0 2 -1 2;1 4 -7 6];
b=[8;9;-5;0];
//Решение системы: x=A-1*b
x=inv(A)*b
//проверка: A*x=b
A*x
```

Листинг 3.33

Результаты работы S–файла

```
--> x =
    3.
   -4.
   -1.
    1.
ans =
    8.
    9.
   -5.
  4.219D-15
```

Листинг 3.34

ЗАДАЧА 4. Решить систему линейных уравнений методом Гаусса:

$$2x_1 - x_2 + 5x_3 = 0,$$

$$3x_1 + 2x_2 - 5x_3 = 1,$$

$$x_1 + x_2 - 2x_3 = 4.$$

Решение системы линейных уравнений при помощи *метода Гаусса* основывается на том, что от заданной системы, переходят к эквивалентной системе, которая решается проще, чем исходная система.

Метод Гаусса состоит из двух этапов. Первый этап это *прямой ход*, в результате которого расширенная матрица⁶ системы путем элементарных преобразований (перестановка уравнений системы, умножение уравнений на число отличное от нуля и сложение уравнений) приводится к ступенчатому виду. На втором этапе (*обратный ход*) ступенчатую матрицу преобразовывают так, чтобы в первых n столбцах получилась единичная матрица. Последний, $n+1$ столбец этой матрицы содержит решение системы линейных уравнений. Листинг 3.35 представляет текст файла–сценария, а листинг 3.36 результат работы S–файла.

⁵ $A \cdot x = B \Rightarrow A^{-1} A x = A^{-1} B$, т.к. $A^{-1} A = E$, а $E \cdot x = x$, то $x = A^{-1} B$.

⁶ Матрица $(A|b)$, которая формируется путем приписывания к матрице коэффициентов A столбца свободных членов b , называется *расширенной матрицей* системы.

```
//Решение системы методом Гаусса
A=[2 -1 1;3 2 -5;1 3 -2];
b=[0;1;4];
//Приведение расширенной матрицы к треугольному виду
C=rref([A b]);
//Выделение последнего столбца из матрицы,
//x - решение системы
x=C(1:3,4:4)
A*x //Проверка
```

Листинг 3.35

```
--> x =
    0.4642857
    1.6785714
    0.75
ans =
- 1.110D-15
    1.
    4.
```

Листинг 3.36

4. Построение двумерных графиков

4.1. Функция plot

Для построения графиков вида $y=f(x)$ в Scilab существует функция `plot`. Синтаксис этой функции сильно изменился в четвертой версии пакета, поэтому сначала рассмотрим синтаксис предыдущих версий Scilab, а затем посмотрим что изменилось в четвертой версии.

В предыдущих версиях Scilab обращение к функции `plot` предназначена для построения графика функции $y=f(x)$ (Scilab 3). Обращение к функции имеет вид:

```
plot(x, y, [xcar, ycar, caption])
```

Здесь x – массив абсцисс, y – массив ординат, $xcar$, $ycar$, $caption$ – подписи осей X , Y и графика соответственно. Рассмотрим ее использование на примере построения функции $y=\sin(\cos(x))$.

```
x=-2*%pi:0.1:2*%pi;
y=sin(cos(x));
plot(x,y,'X','Y','plot function y=sin(cos(x))');
```

Листинг 4.1.

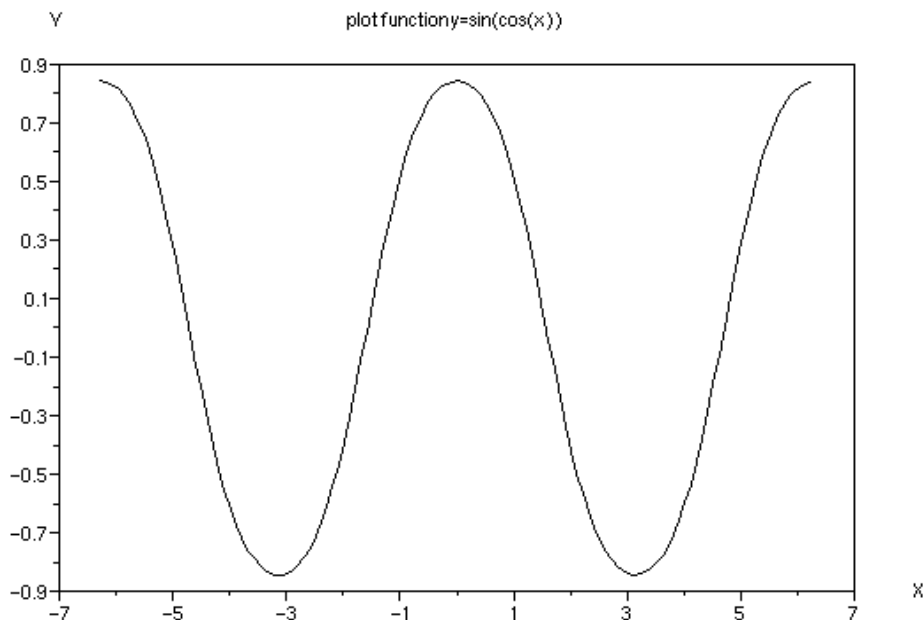


Рис. 4.1. График функции $y=\sin(\cos(x))$

В простейшем случае обращение к функции имеет вид `plot(y)`, в качестве массива x выступает массив номеров точек массива y . В этом случае с помощью функции можно построить графики нескольких функций (см. листинг 4.2 и рис. 4.2).

```
x=-2*%pi:0.1:2*%pi;
plot([sin(cos(x));cos(sin(x));exp(sin(x));exp(cos(x))]);
```

Листинг 4.2.

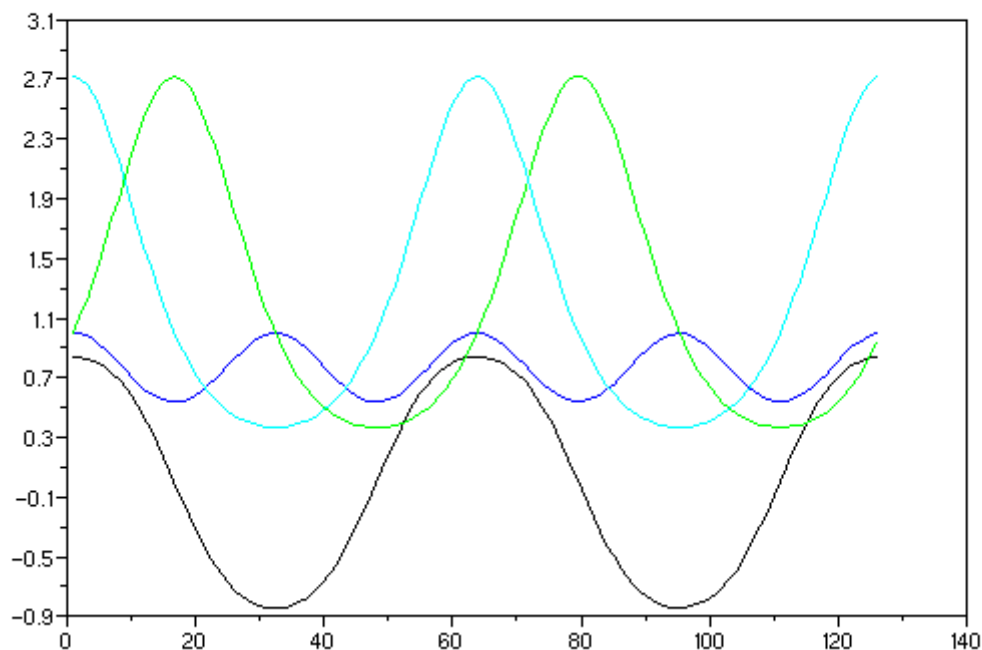


Рис. 4.2. Графики функций $y=\sin(\cos(x))$, $z=\cos(\sin(x))$, $u=e^{\sin(x)}$, $v=e^{\cos(x)}$

Как видно, из рис. 4.2, в Scilab 3 функция `plot` не позволяет построить полноценные графики нескольких функций.

Поэтому в Scilab 4.0 функция `plot` значительно модифицирована и ее возможности теперь значительно расширены и сопоставимы с возможностями функции `plot` из Matlab 7. При простейшем обращении к функции `plot(x, y)` будет создано окно с именем `Scilab Graphic (0)`, в котором будет построен график функции $y(x)$ на интервале (см. на рис. 6.3 график функции $y=\sin(\cos(x))$).

Если повторно обратиться к функции `plot`, то будет создано новое графическое окно и в нем будет построен новый график. Для построения нескольких графиков в одной системе координат можно поступить одним из следующих способов:

1. Обратиться к функции `plot` следующим образом `plot(x1, y1, x2, y2, ..., xn, yn)`, где x_1, y_1 – массивы абсцисс и ординат первого графика; x_2, y_2 – массивы абсцисс и ординат второго графика; ..., x_n, y_n – массивы абсцисс и ординат n -ого графика. Например

```
x=-6.28:0.02:6.28;
y=sin(x/2);
z=cos(x);
v=exp(cos(x));
plot(x, y, x, z, x, v);
```

Листинг 4.4.

Полученный график представлен на рис. 4.4.

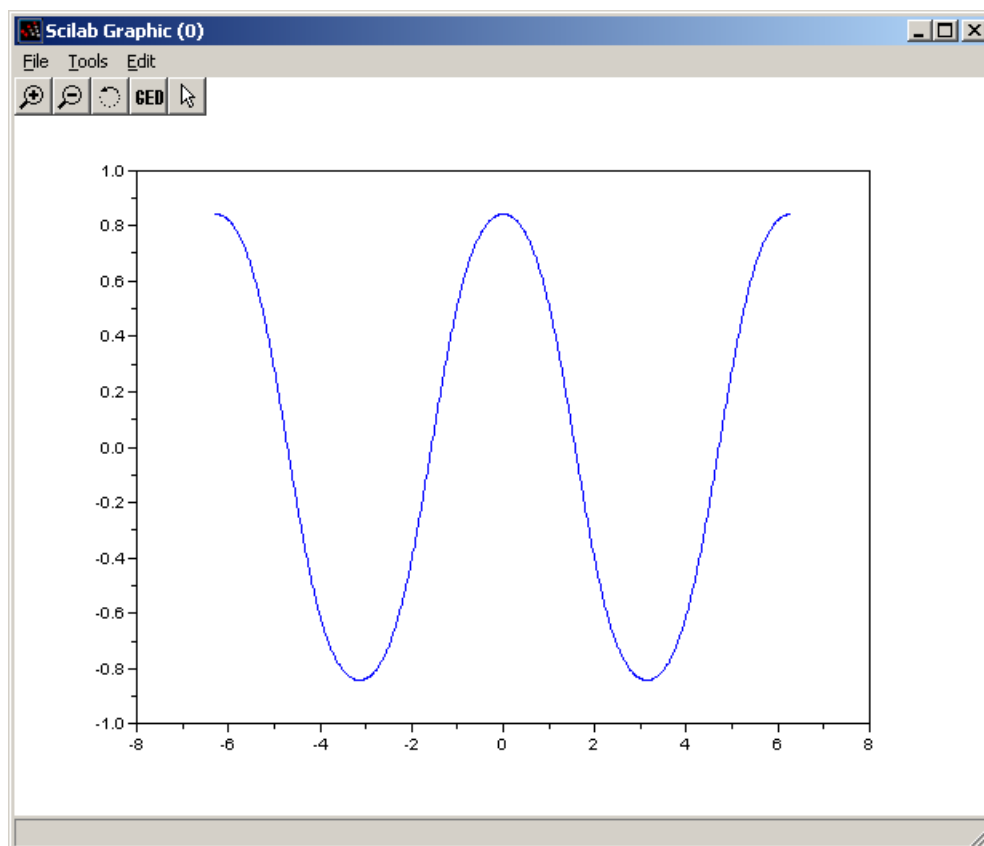


Рис. 4.3: График функции $y = \sin(\cos(x))$

2. Каждый график изображать с помощью функции `plot(x,y)`, но перед обращением к функциям `plot(x2,y2)`, `plot(x3,y3)`, ..., `plot(xn,yn)` вызвать команду `mtlb_hold(' on')`, которая блокирует режим создания нового окна.

```
x=-6.28:0.02:6.28;
y=sin(x/2);
plot(x,y);
mtlb_hold(' on');
z=cos(x);
plot(x,z);
mtlb_hold(' on');
v=exp(cos(x));
plot(x,v);
```

Листинг 4.5

Обратите внимание, что при построении графиков первым способом Scilab автоматически изменяет цвета изображаемых в одной системе координат графиков. Однако управлять цветом и видом каждого из изображаемых графиков может и пользователь, для чего необходимо воспользоваться полной формой функции `plot`:

```
plot(x1, y1, s1, x2, y2, s2, ..., xn, yn, sn)
```

где x_1, x_2, \dots, x_n – массивы абсцисс графиков; y_1, y_2, \dots, y_n – массивы ординат графиков; s_1, s_2, \dots, s_n – строка, состоящая из трех символов, которые определяют цвет линии, тип маркера и тип линии графиков (см. табл. 4.1-4.3), в строке могут использоваться один или два символа.

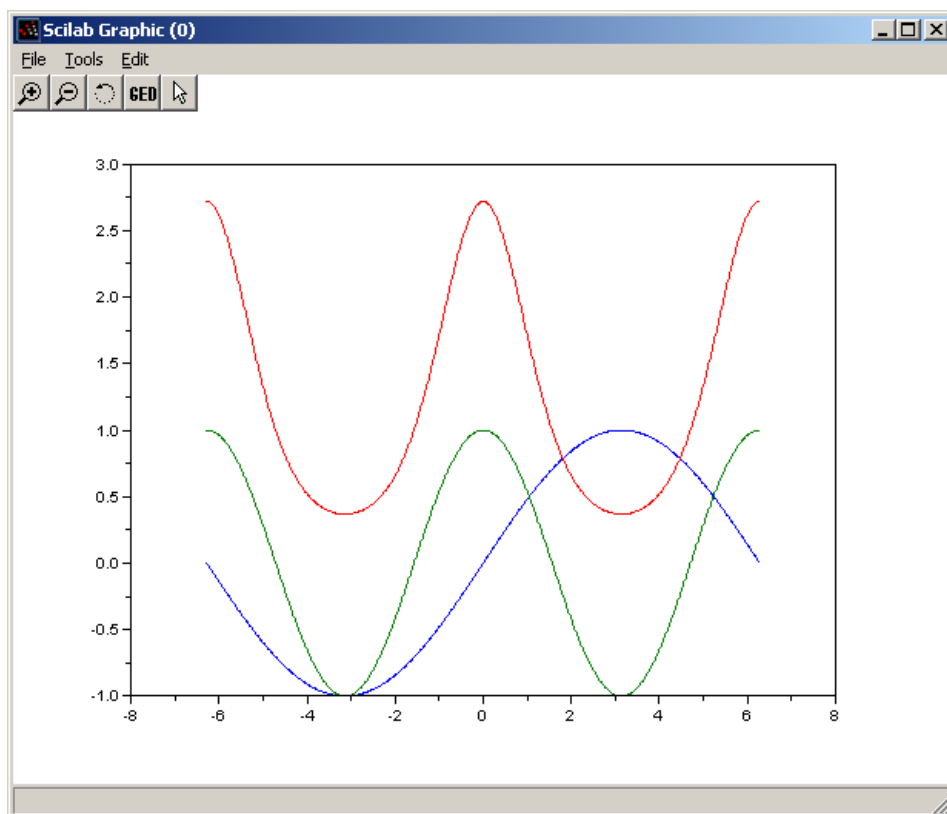


Рис. 4.4: График функции $y=\sin(x/2)$, $z=\cos(x)$; $v=\exp(\cos(x))$ в Scilab 4

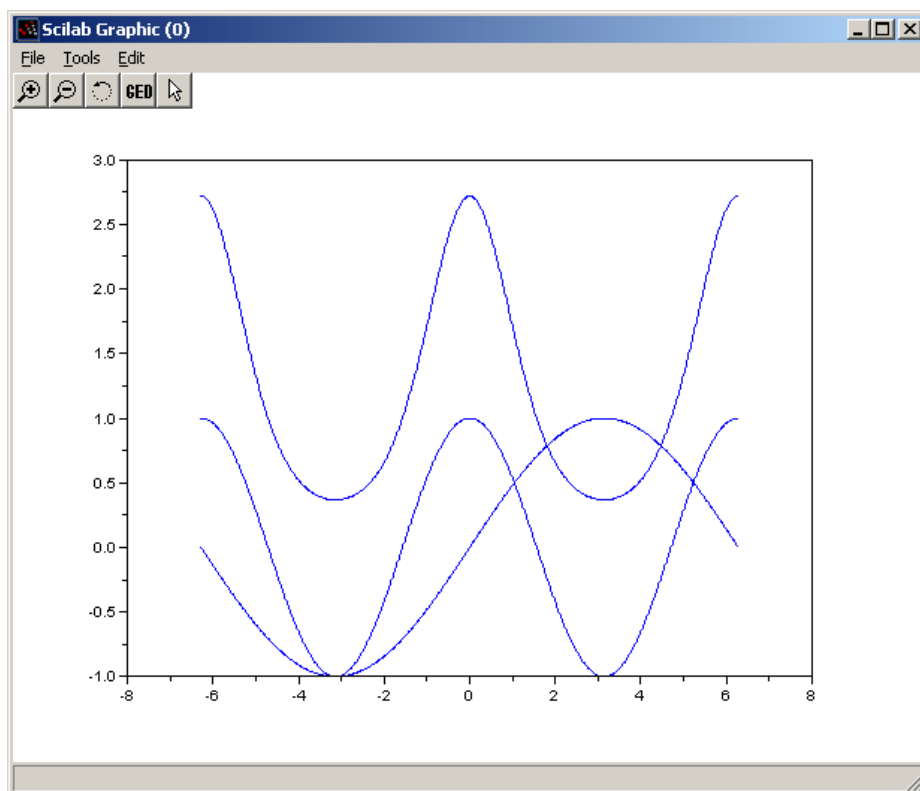


Рис. 4.5: Построение графиков функций $y=\sin(x/2)$, $z=\cos(x)$; $v=\exp(\cos(x))$ в Scilab 4 вторым способом

Таблица 4.1. Символы цветов линий

Символ	Описание
y	желтый
m	розовый
c	голубой
r	красный
g	зеленый
b	синий
w	белый
k	черный

Таблица 4.2. Символы типов маркера

Символ	Описание
.	точка
o	кружок
x	крестик
+	знак "плюс"
*	звездочка
s	квадрат
d	ромб
v	треугольник вершиной вниз
^	треугольник вершиной вверх
<	треугольник вершиной влево
>	треугольник вершиной вправо
p	пятиконечная звезда
h	шестиконечная звезда

Таблица 4.3. Символы типов линий

Символ	Описание
-	сплошная
:	пунктирная
-.	штрихпунктирная
--	штриховая

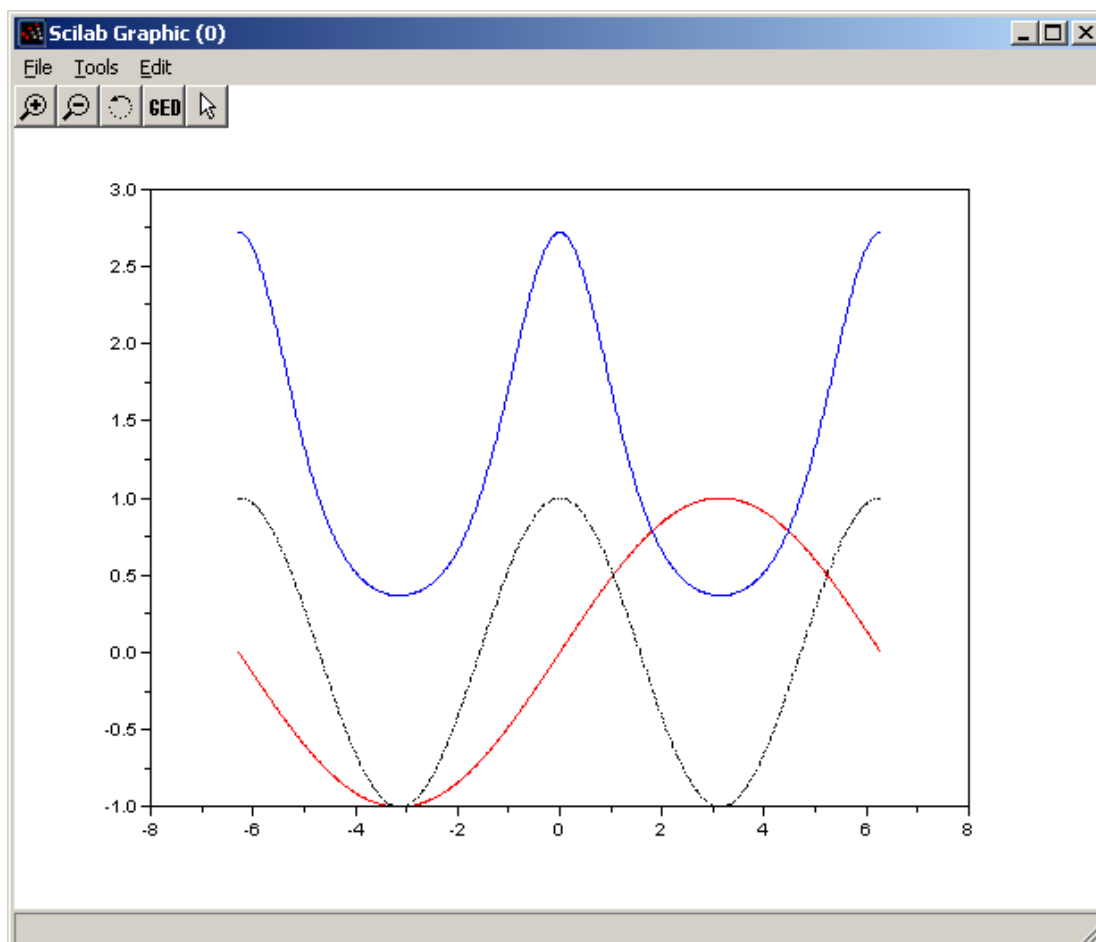
В качестве примера рассмотрим построение графиков функций $y=\sin(x/2)$, $z=\cos(x)$, $v=\exp(\cos(2x))$ на интервале с использованием управлением свойствами линий, в результате чего формируется окно, представленного на рис. 4.6.

```
x=-6.28:0.02:6.28;
y=sin(x/2);
plot(x,y,'r-');
mtlb_hold('on');
```

```

z=cos(x);
plot(x,z,'k:');
mtlb_hold('on');
v=exp(cos(x));
plot(x,v,'b-');

```

Листинг 4.6*Рис. 4.6: Построение графиков в Scilab с указанием свойств линий*

Следующей функцией, которая может быть использована для построения графиков, является функция `plot2d`.

4.2. Функция `plot2d`

В общем виде обращение к функции имеет вид:

```
plot2d([loglog],x,y,[key1=value1,key2=value2,...,keyn=valuen])
```

- `logflag` – строка из двух символов, каждый из которых определяет тип осей (`n` – нормальная ось, `l` – логарифмическая ось), по умолчанию "nn";
- `x` – массив абсцисс;
- `y` – массив ординат (или матрица, каждый столбец которого содержит массив ординат очередного графика) (количество элементов в массиве `x` и `y` должно быть одинаковым), если `x` и `y` – являются матрицами одного размера, то в этом случае, каждый столбец матрицы `y` отображается относительно соответствующего столбца

матрицы x ;

- $key_i=value_i$ – последовательность значений параметров графиков, возможны следующие значения параметров: `style` – определяет массив (`mas`) числовых значений цветов графика (`id` цвета), количество элементов массива совпадает с количеством изображаемых графиков, по умолчанию, по умолчанию представляет собой массив $mas_i=i$, цвет i -й линии совпадает с номером i , для формирования `id` соответствующего цвета (кода цвета) можно воспользоваться функцией `color`, которая по названию (`color("цвет")`) или коду `grb` (`color(r,g,b)`) цвета формирует нужный `id` (код) цвета. Если значение стиля отрицательное то это будет точечный график без соединения точек между собой линиями. Пример построения нескольких графиков различного цвета приведен ниже (см. листинг 4.7 и рис.4.7).

```
x=[-2*%pi:0.1:2*%pi];  
y=[sin(x); cos(x)];  
plot2d(x,y',style=[color("red"), color("blue")]);
```

Листинг 4.7.

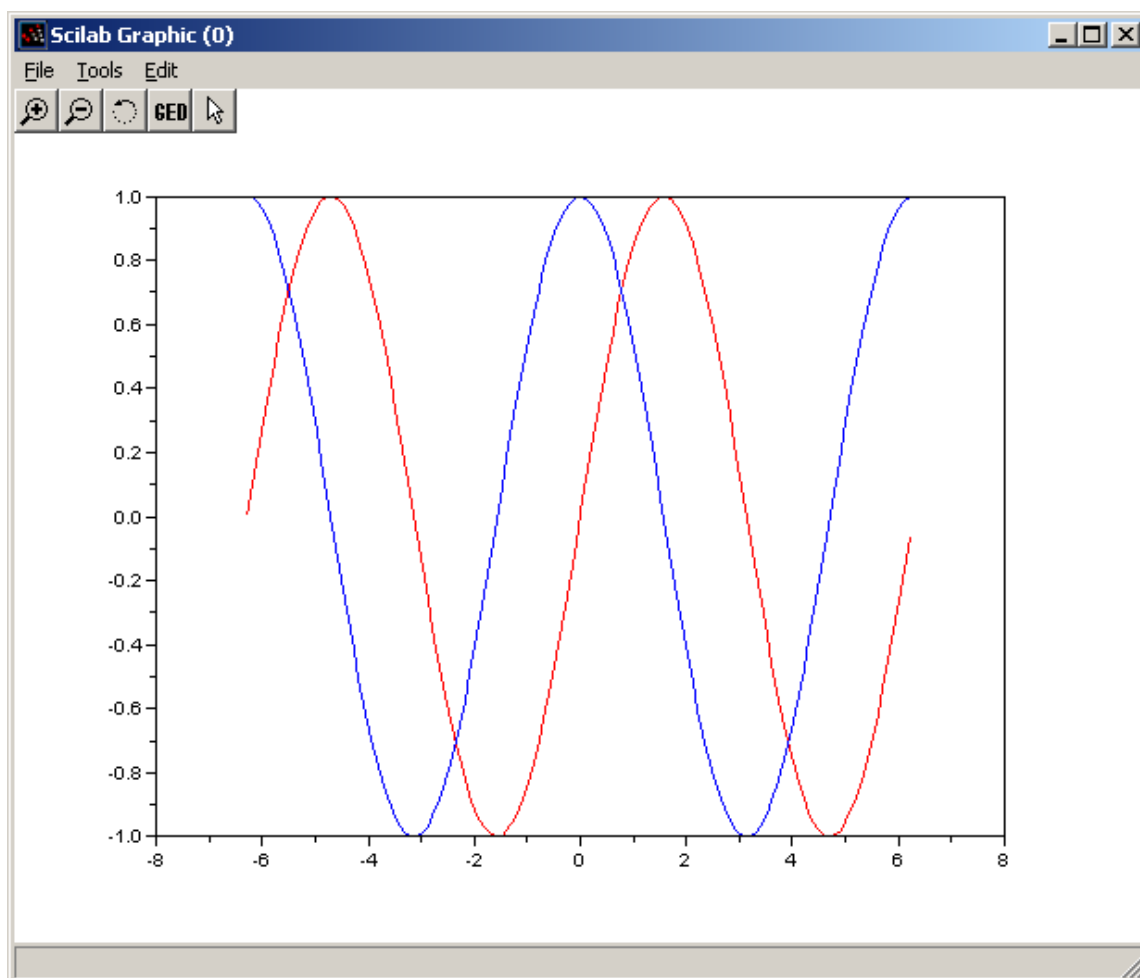


Рис. 4.7. Использование параметра `style` в функции `plot2d`

- `rect` – этот вектор [`xmin`, `ymin`, `xmax`, `ymax`] определяет размер окна вокруг графика, пример использования этого параметра приведен на листинге 4.8 и

рис. 4.8;

```
x=[-2*%pi:0.1:2*%pi];
y=[sin(x); cos(x)];
plot2d(x,y',style=[color("red"),color("blue")],rect=[-8,-2,8,2]);
```

Листинг 4.8.

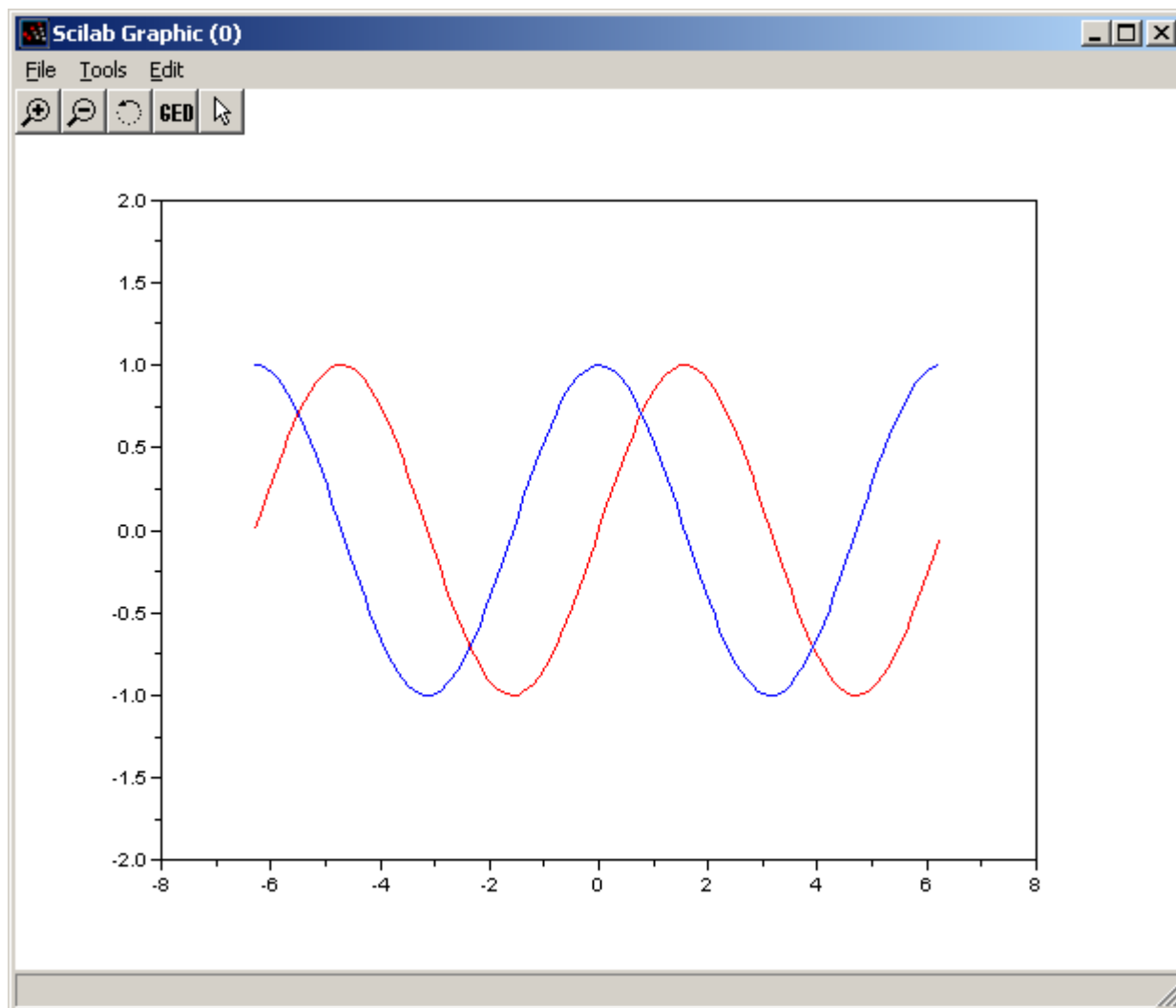


Рис. 4.8. Использование параметра *rect* в функции *plot2d*

- *frameflag* – параметр определяет окно в котором, будет изображаться график, он может принимать следующие значения: 0 – не вычислять размеры окна, использовать значения по умолчанию или значения из предыдущего графика, 1 – размер окна определяется параметром *rect*, 2 – размер окна определяется из соотношения между минимальным или максимальным значениями *x* и *y*, 3 – размер окна определяется параметром *rect* в в изометрическом масштабе, 4– размер окна определяется из соотношения между минимальным или максимальным значениями *x* и *y* в изометрическом масштабе,

- *axesflag* - параметр, который определяет рамку вокруг графика, следует выделить следующие значения этого параметра: 0 – нет рамки вокруг графика (см. листинг 4.9 и рис. 4.9); 1 – изображение рамки, ось *y* слева (см. рис. 4.10); 3 –

изображение рамки, ось у справа (см. рис. 4.11); 5 – изображение осей проходящих через точку (0,0) (см. рис. 4.12);

```
x=[-2*%pi:0.1:2*%pi];  
y=[sin(x); cos(x)];  
plot2d(x,y',style=[color("red"), color("blue")], axesflag=0);
```

Листинг 4.9.

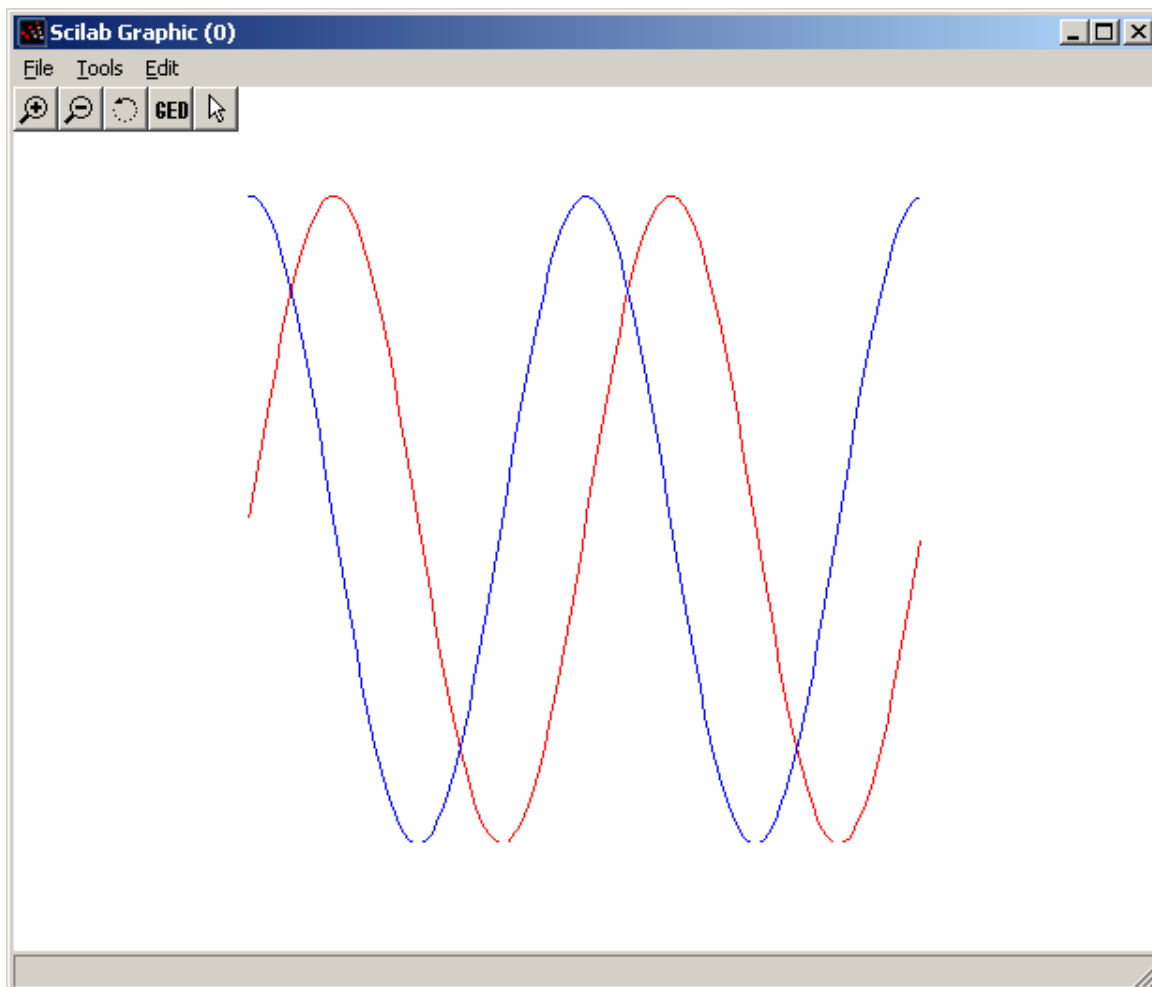


Рис. 4.9. Параметр axesflag=0 в функции plot2d

- `naх` – этот параметр используют, если параметр `axesflag` равен 1, `naх` представляет массив из четырех значений `[nx, Nx, ny, Ny]` – где `Nx` (`Ny`) – число основных делений с подписями под осью `X` (`Y`), `nx` (`ny`) – число промежуточных делений;
- `leg` – строка, определяющая легенды для каждого графика, структура строки такая: `"leg1@leg2@leg3@...@legn"`, где `leg1` – легенда первого графика, ..., `legn` – легенда первого графика.

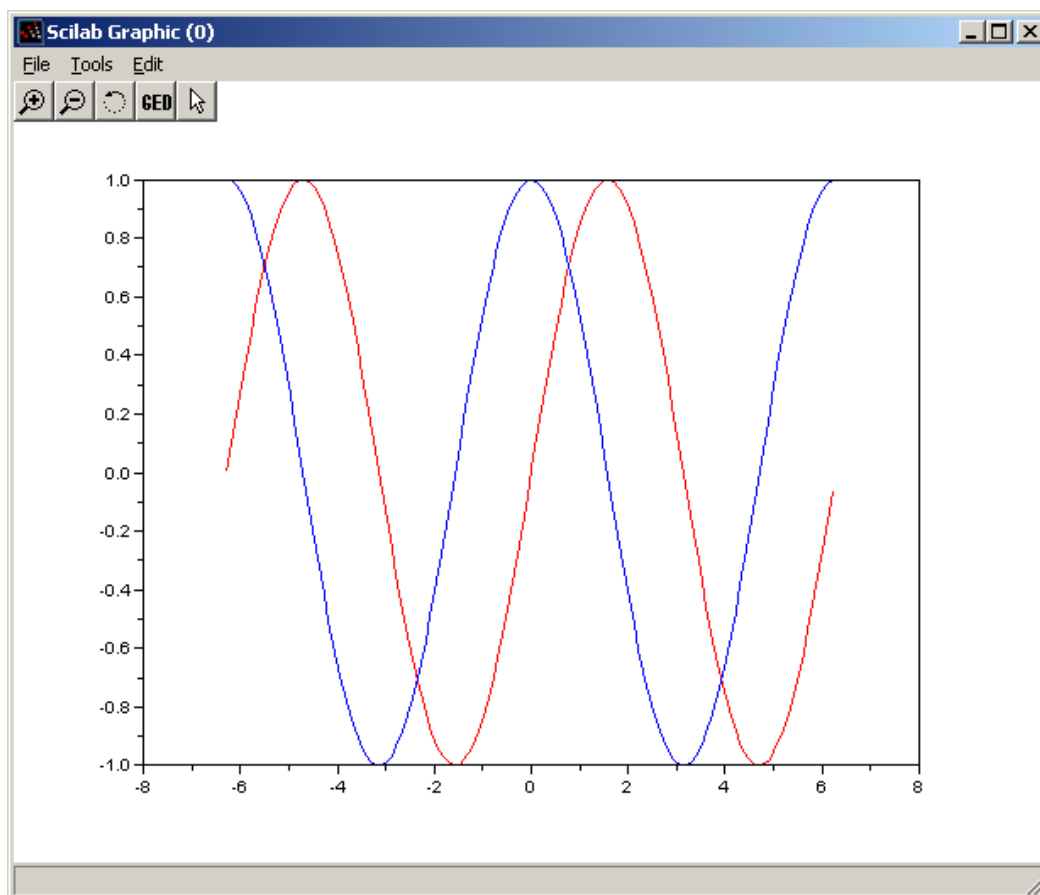


Рис. 4.10. Параметр `axesflag=1` в функции `plot2d`

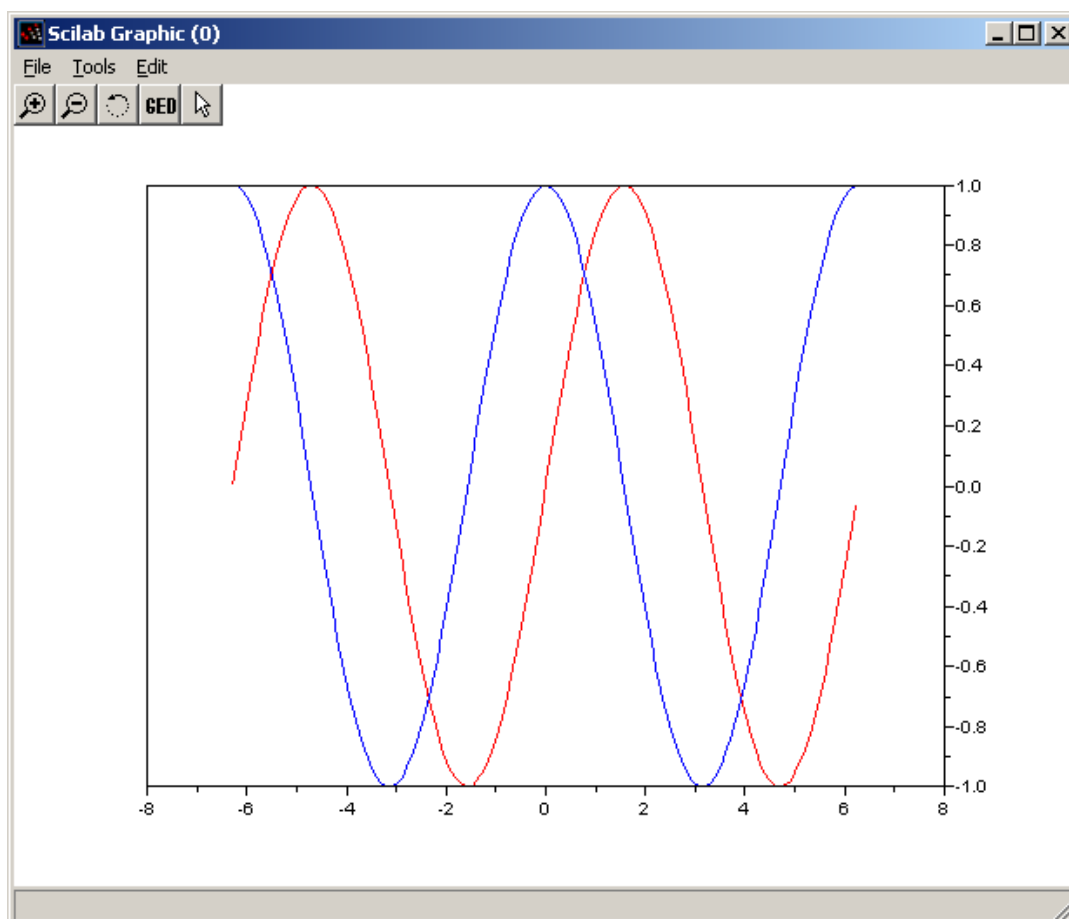


Рис. 4.11. Параметр `axesflag=3` в функции `plot2d`

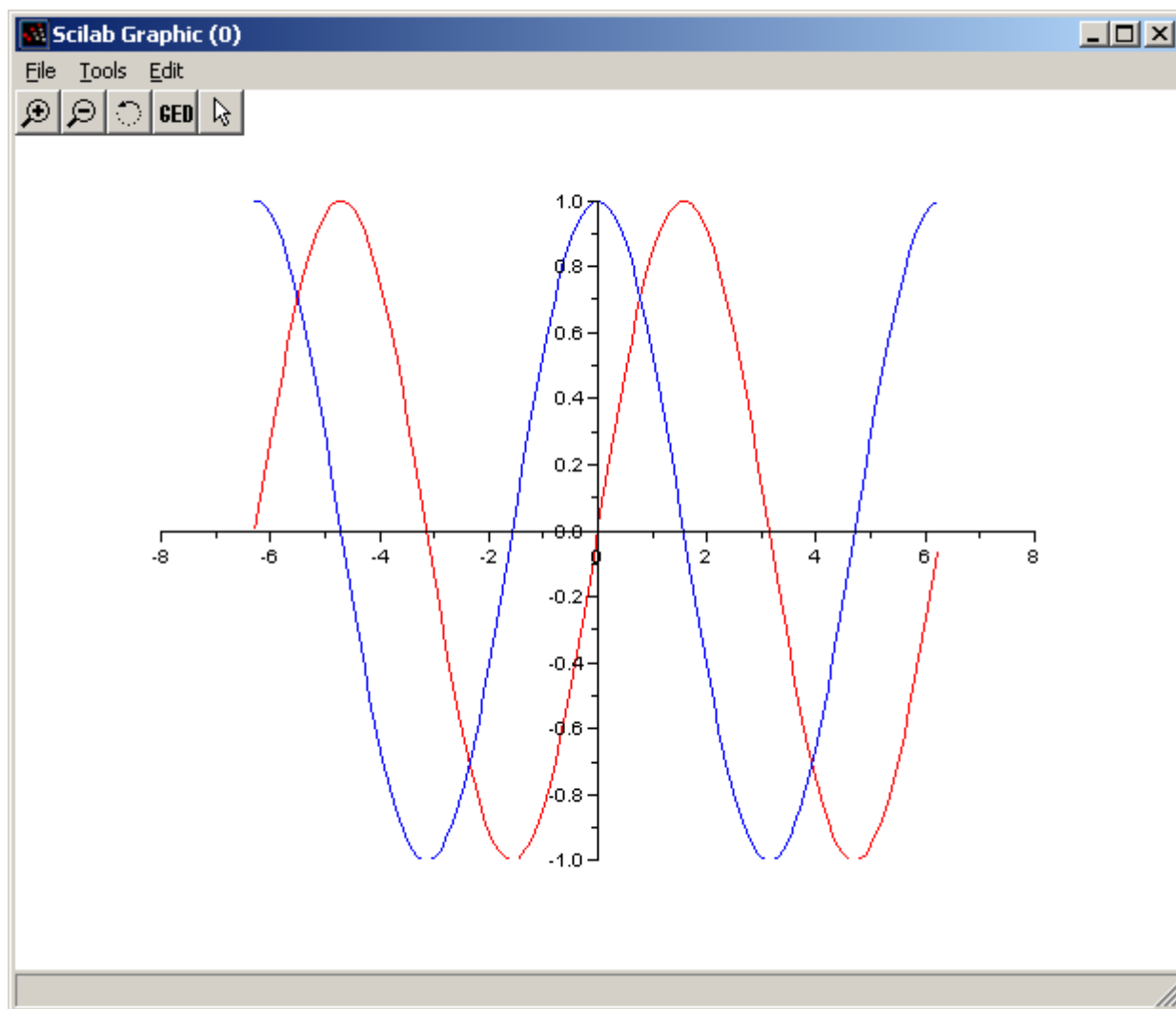


Рис. 4.12. Параметр `axesflag=5` в функции `plot2d`

На листинге 4.8 и рис. 4.13 приведен пример построения графиков функций с использованием параметра `naх` при построении функции `plot2d`.

```
x=[-8:0.1:8];
y=[sin(x); cos(x)];
plot2d(x,y',style=[color("red"),color("blue")],axesflag=1,
naх=[4,9,3,6]);
```

Листинг 4.9

На листинге 4.9 приведен пример построения графиков функции с использованием легенд.

```
x=[-2*pi:0.1:2*pi];
y=[sin(x); cos(x)];
plot2d(x,y',style=[color("red"), color("blue")], axesflag=5,
leg="sin(x)@cos(x)");
```

Листинг 4.10.

Функцию `plot2d` можно использовать для построения точечных графиков. В этом случае обращение к функции имеет вид

```
plot2d(x,y,d),
```

здесь `d` – отрицательное число, определяющее тип маркера (см. листинг 4.10 и рис. 4.14).

```
x=[-2*%pi:0.25:2*%pi];  
y=sin(x);  
plot2d(x,y,-3);
```

ЛИСТИНГ 4.10

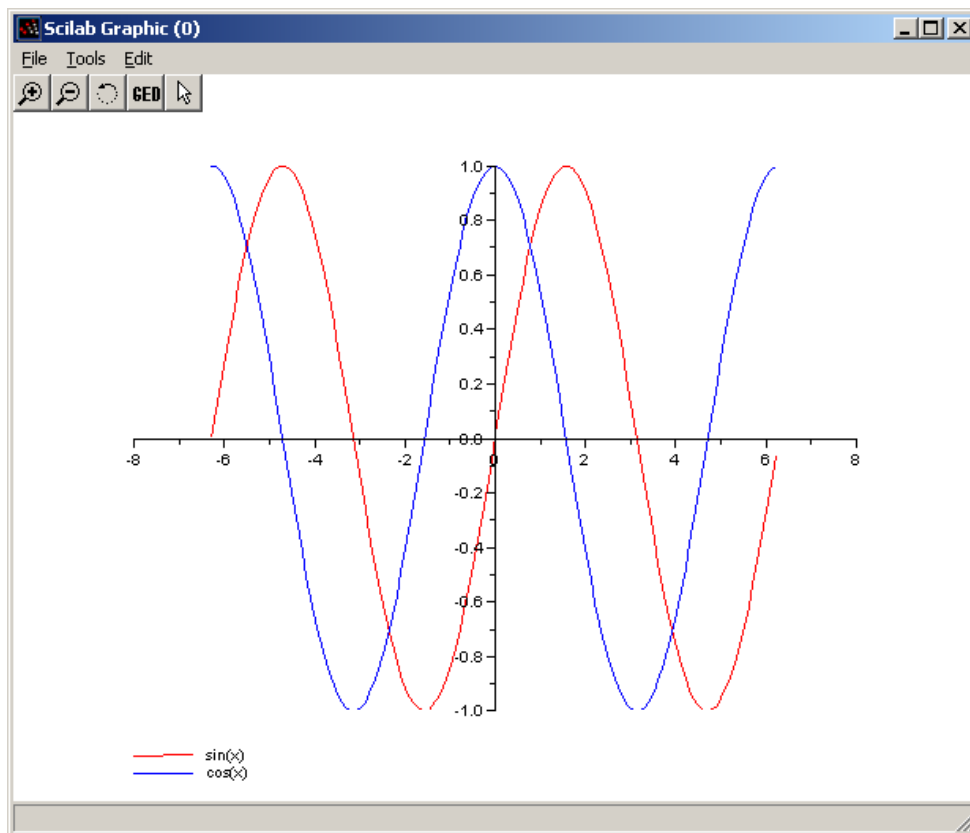


Рис. 4.13. Использование параметра *leg* в функции *plot2d*

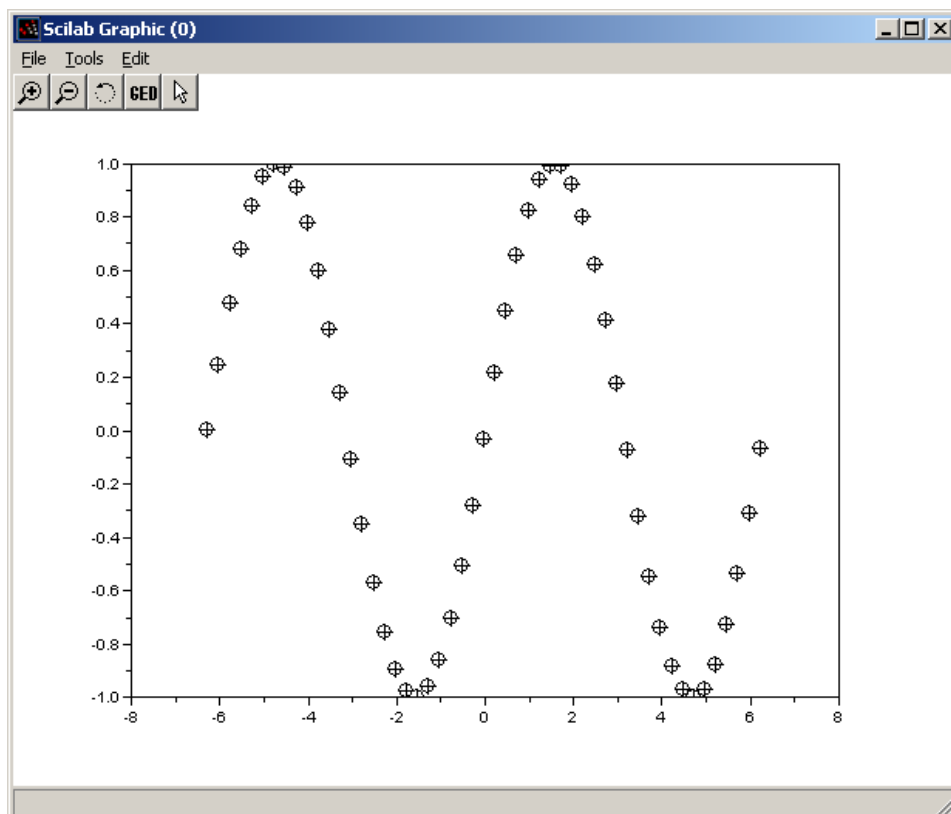


Рис. 4.14. Построение точечного графика

Для изображения графика в виде ступенчатой линии в Scilab есть функция `plot2d2(x, y)`.

Пример ее использования приведен на листинге 4.11 и на рис. 4.15.

```
x=[1911,1941,1961,1981,1991,1996];
y=[20,300,350,1100,1030,1020];
plot2d2(x, y);
```

Листинг 4.11.

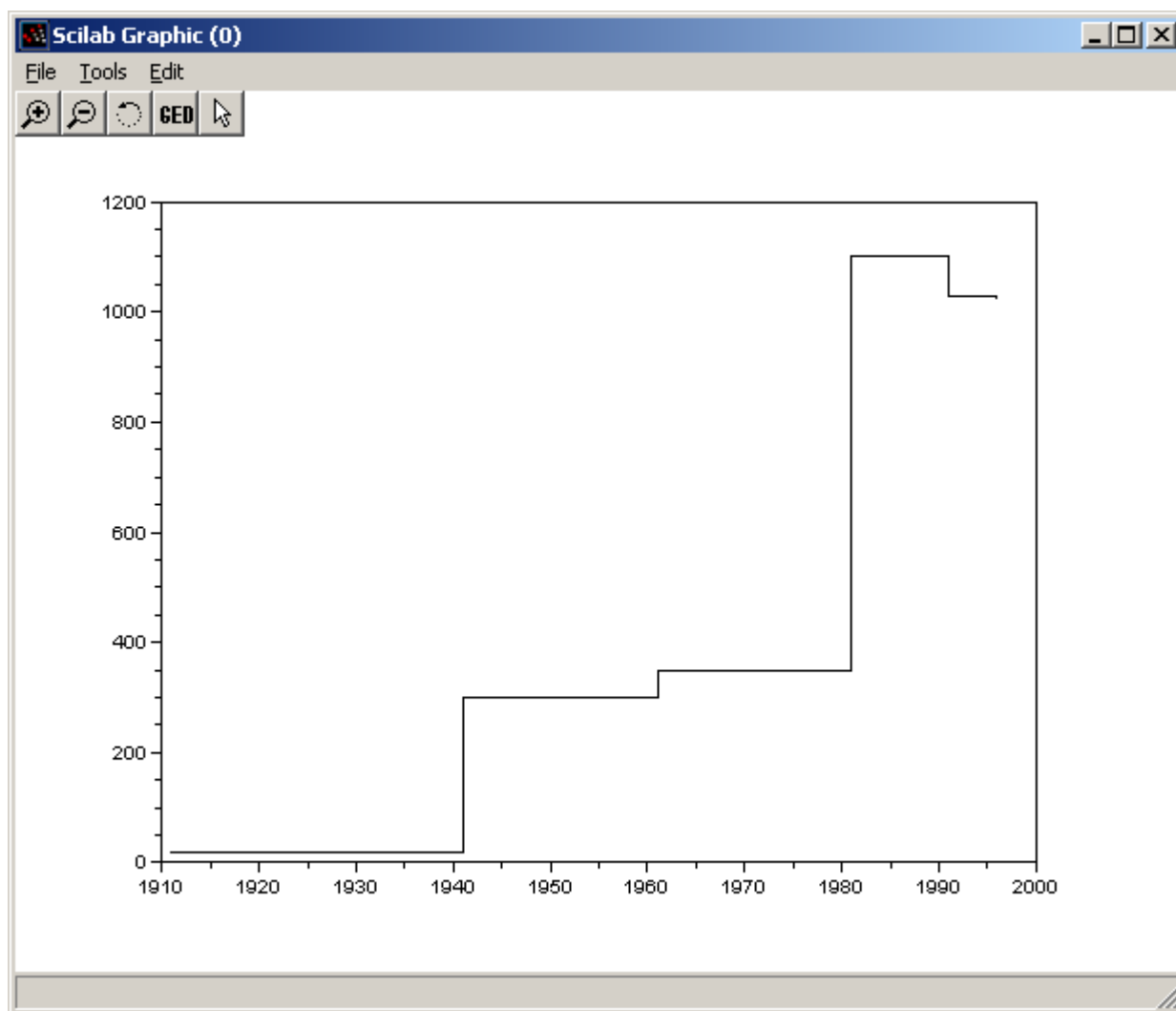


Рис. 4.15. Пример изображения графика в виде ступенчатой линии

4.3. Построение полярных графиков

Для построения графиков в полярной системе координат в Scilab служит функция `polarplot`

```
polarplot(fi, ro, [key1=value1, key2=value2, ..., keyn=valuen])
```

Здесь fi - полярный угол, ro - полярный радиус.

Рассмотрим пример построения полярных графиков $\rho = 3\cos(5\varphi)$, $\rho_1 = 3\cos(3\varphi)$ (см. листинг 4.12, рис. 4.15).

```
fi=0:0.01:2*%pi;
```

```

ro=3*cos(5*fi);
ro1=3*cos(3*fi);
polarplot(fi,ro,style=color("red"));
polarplot(fi,ro1,style=color("blue"));

```

Листинг 4.12.

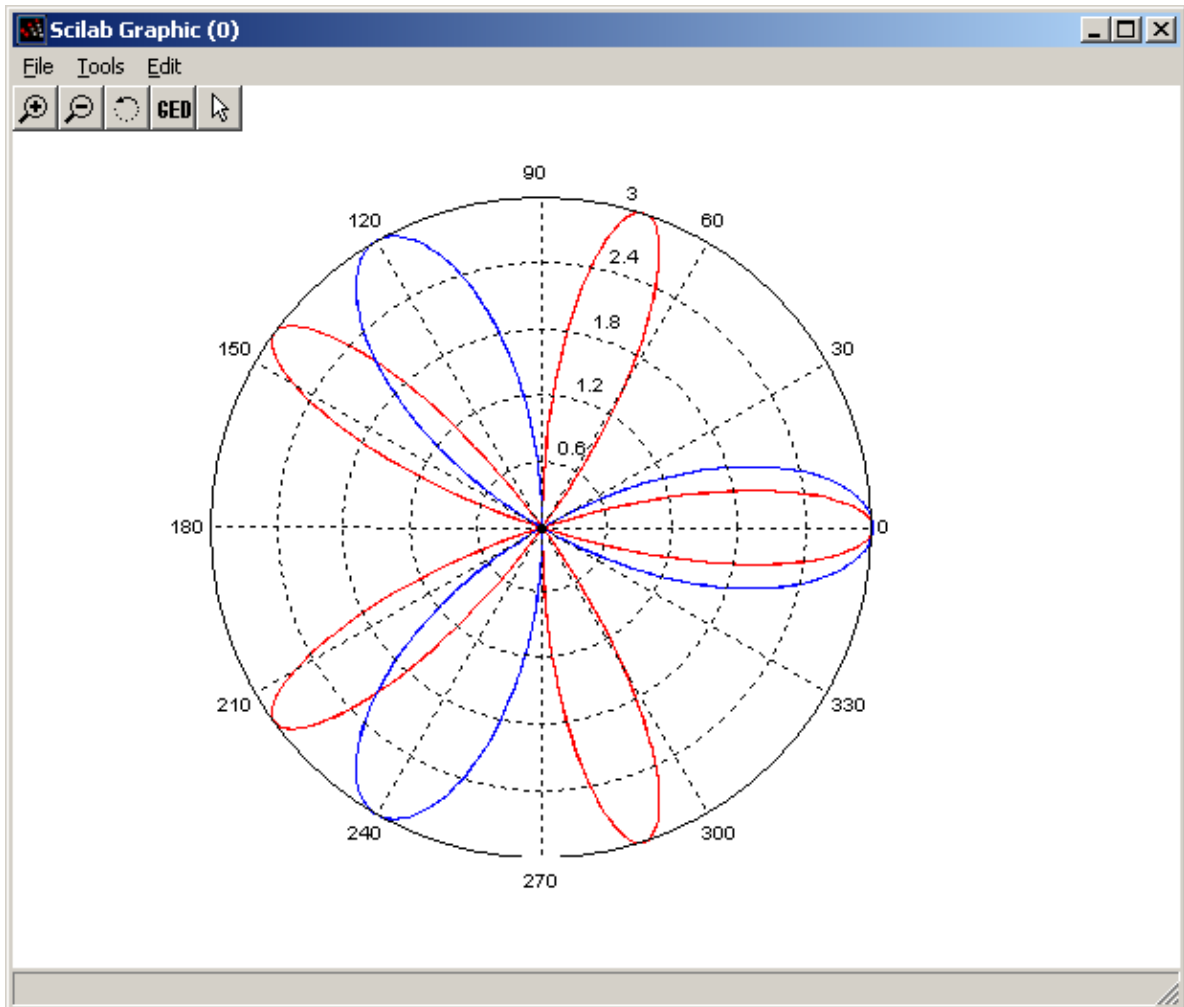


Рис. 4.16. Пример построения полярных графиков

4.4. Построение графиков в параметрической форме

Для построения графиков в параметрической форме можно воспользоваться функциями **plot2d** или **plot**. В качестве примера параметрического графика рассмотрим построение

графика строфоиды

$$\begin{aligned}
 x(t) &= \frac{t^2 - 1}{t^2 + 1} \\
 y(t) &= t \frac{t^2 - 1}{t^2 + 1}
 \end{aligned}
 \quad t = -5, \dots, 5 \quad (\text{см. листинг 4.13, рис. 4.17})$$

```

t=-5:0.01:5;
x=(t.^2-1)./(t.^2+1);
y=t.*(t.^2-1)./(t.^2+1);
plot(x,y);

```

Листинг 4.13

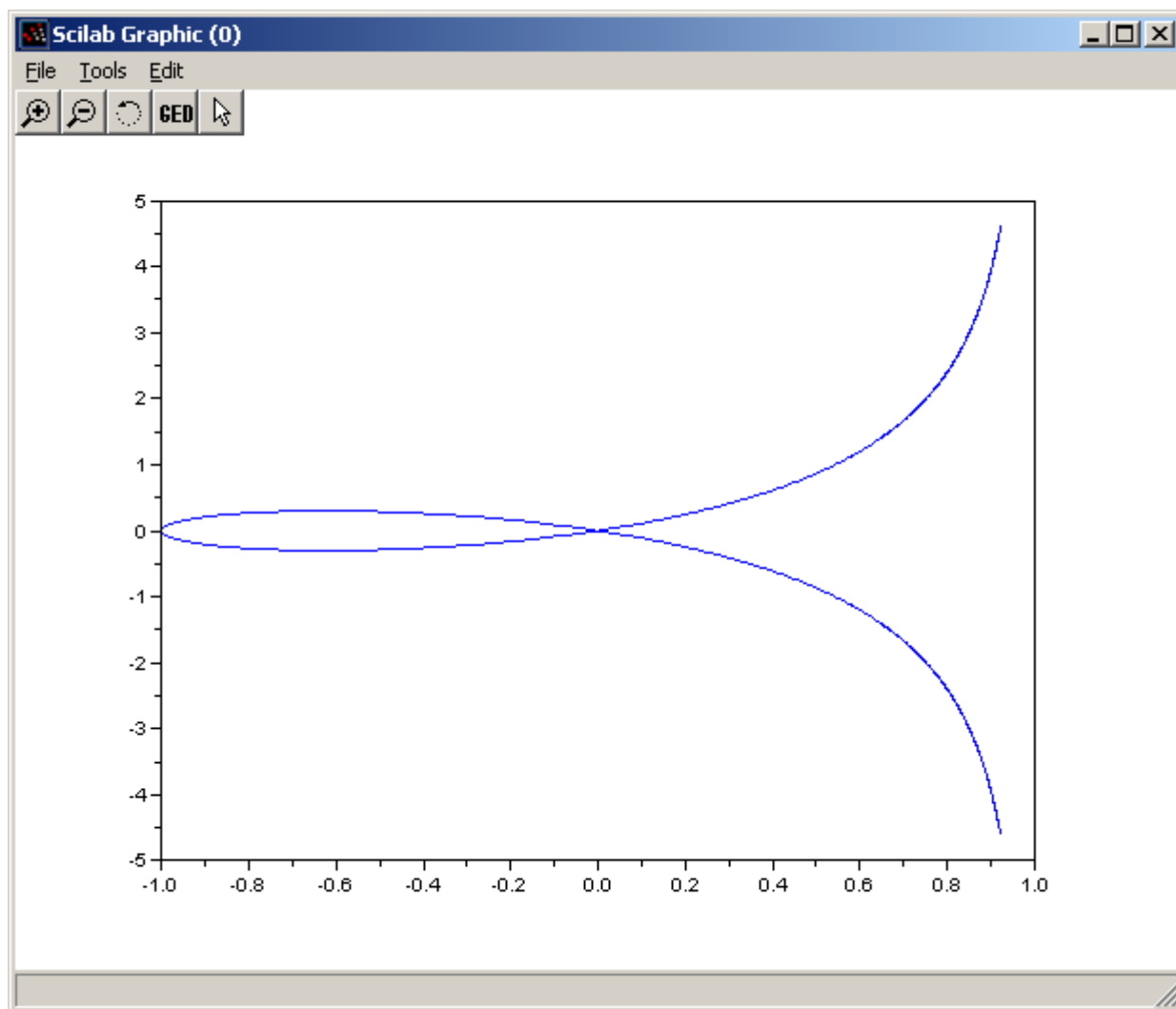


Рис. 4.17. График строфоиды

В качестве еще одного примера параметрического графика рассмотрим полукубическую

параболу $x(t)=0.5t^2$ $y(t)=0.3t^3$ $t=-3 \dots 3$ (см. листинг 4.13, рис. 4.17).

```
t=-3:0.01:3;
x=0.5*t.^2;
y=0.3*t.^3;
plot(x,y);
```

Листинг 4.14.

5. Оформление графиков

Рассмотрим основные возможности **Scilab** по оформлению графиков.

5.1. Изображение сетки на графике

Для изображения сетки следует воспользоваться функцией `xgrid(color)`, `color` определяет id цвета линии сетки.

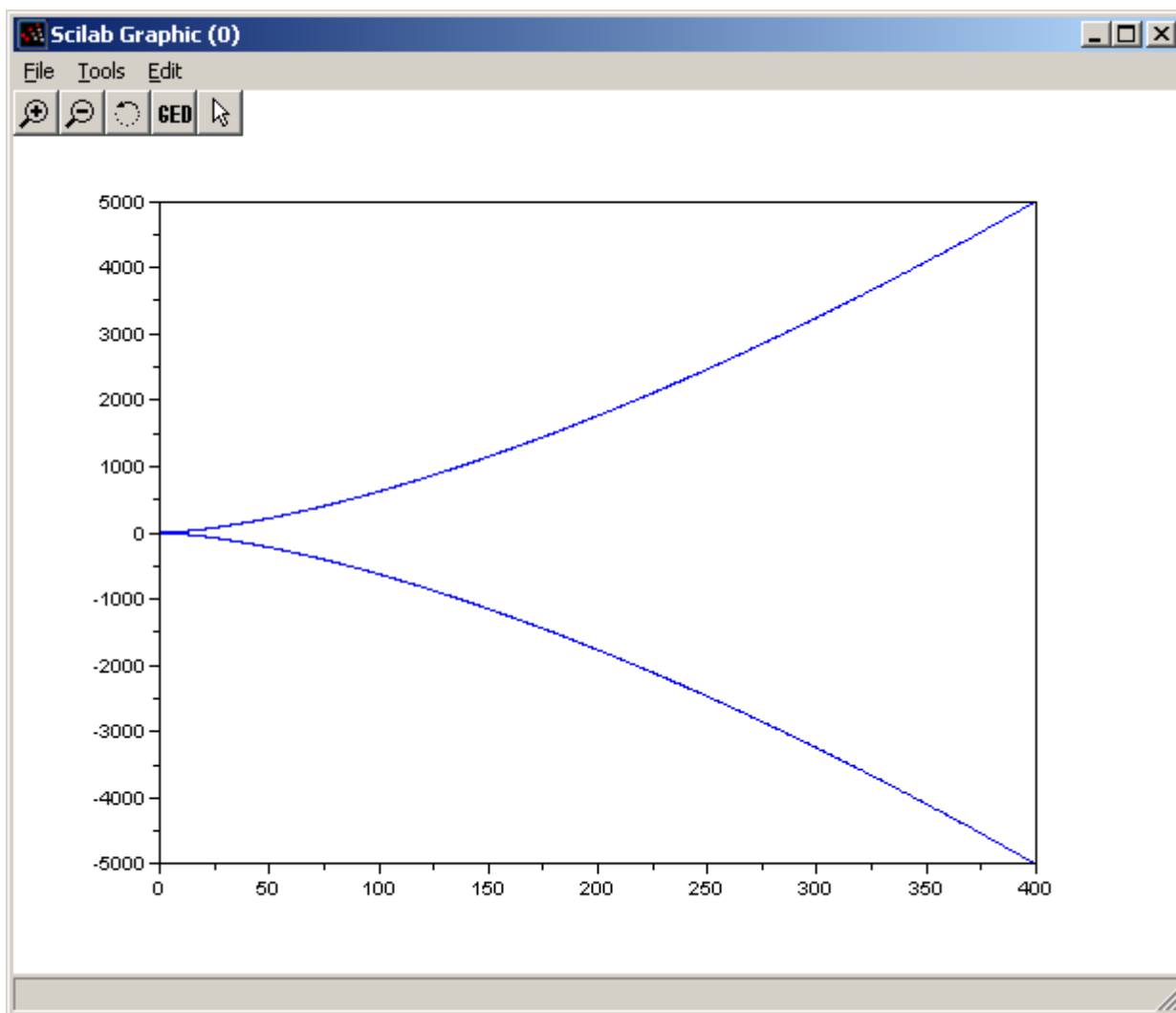


Рис. 4.18. График полукубической параболы

5.2. Заголовки на графике

Для вывода заголовков на графике служит функция

```
xtitle(title, xstr, ystr),
```

Здесь `title` – название графика, `xstr` – название оси X, `ystr` – название оси Y.

5.3. Нанесение легенд на график

Для нанесения легенд на график служит функция

```
legend(leg1, leg2, ..., legn[, pos] [, boxed])
```

`leg1` – имя первой легенды, `leg2` – имя второй легенды, ..., `legn` – имя n-й легенды; `pos` – месторасположение легенды: 1 – верхнем правом углу (по умолчанию), 2 – верхнем левом углу, 3 – нижнем левом углу, 4 – нижнем правом углу, 5 – определяется пользователем после изображения графика; `boxed` – логическая переменная (по умолчанию `%t`), которая определяет рисовать или нет рамку вокруг легенды.

Рассмотри пример оформления графика (см. листинг 4.15, рис. 4.19).

```
x=-10:0.01:10;  
y=sin(cos(x));  
z=cos(sin(x));  
plot(x,y,'r',x,z,'b');  
xgrid();  
xtitle('Grafic y=f(x)', 'X', 'Y');  
legend('sin(cos(x))', 'cos(sin(x))', 3, %f);
```

Листинг 4.15

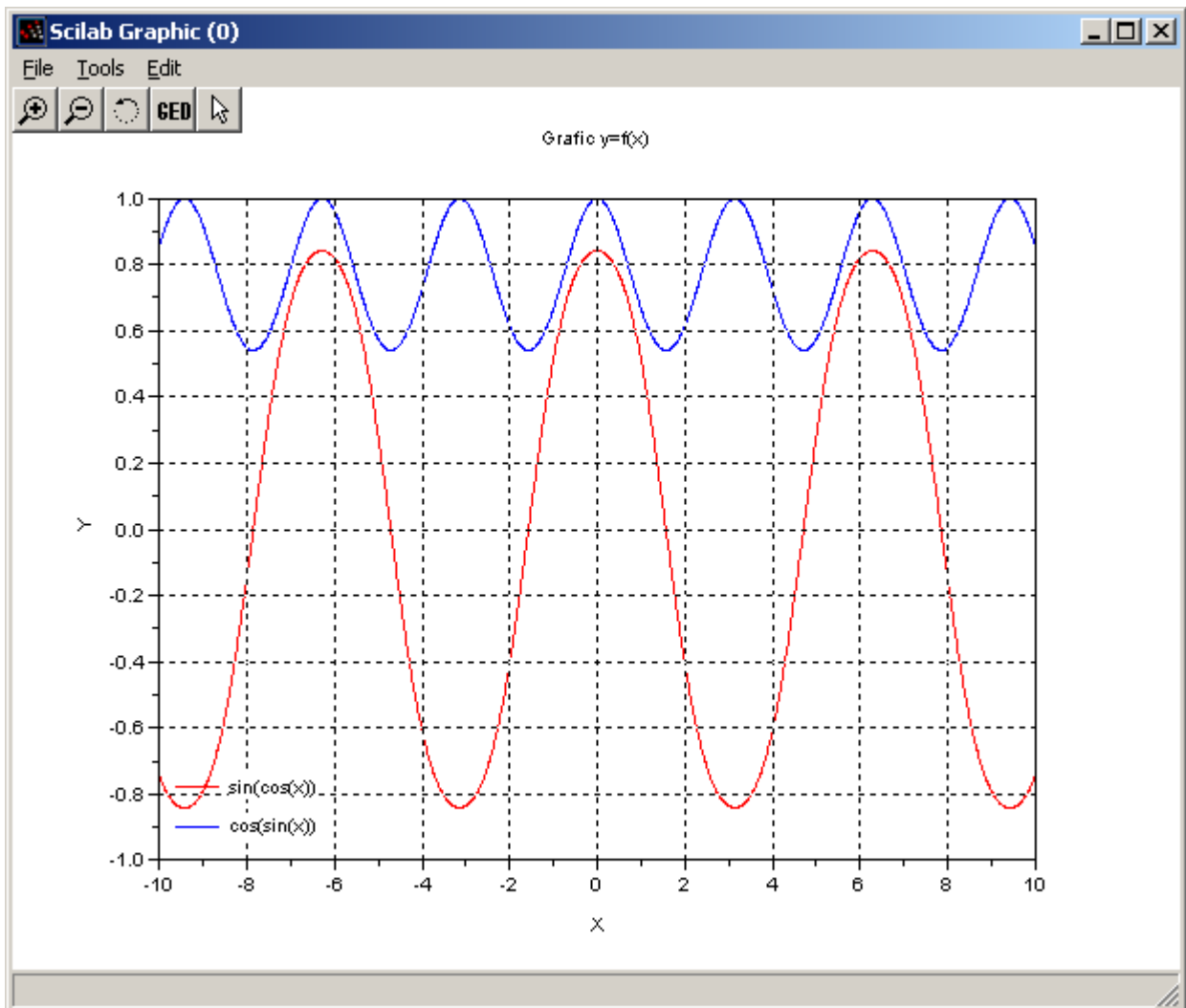


Рис. 4.19. Графики функций $y = \sin(\cos(x))$, $z = \cos(\sin(x))$.

Кроме того, Scilab позволяет после вывода графика перейти в режим форматирования с помощью команды `Edit - Figure properties` в окне график. Окно форматирования графика представлено на рис. 4.20.

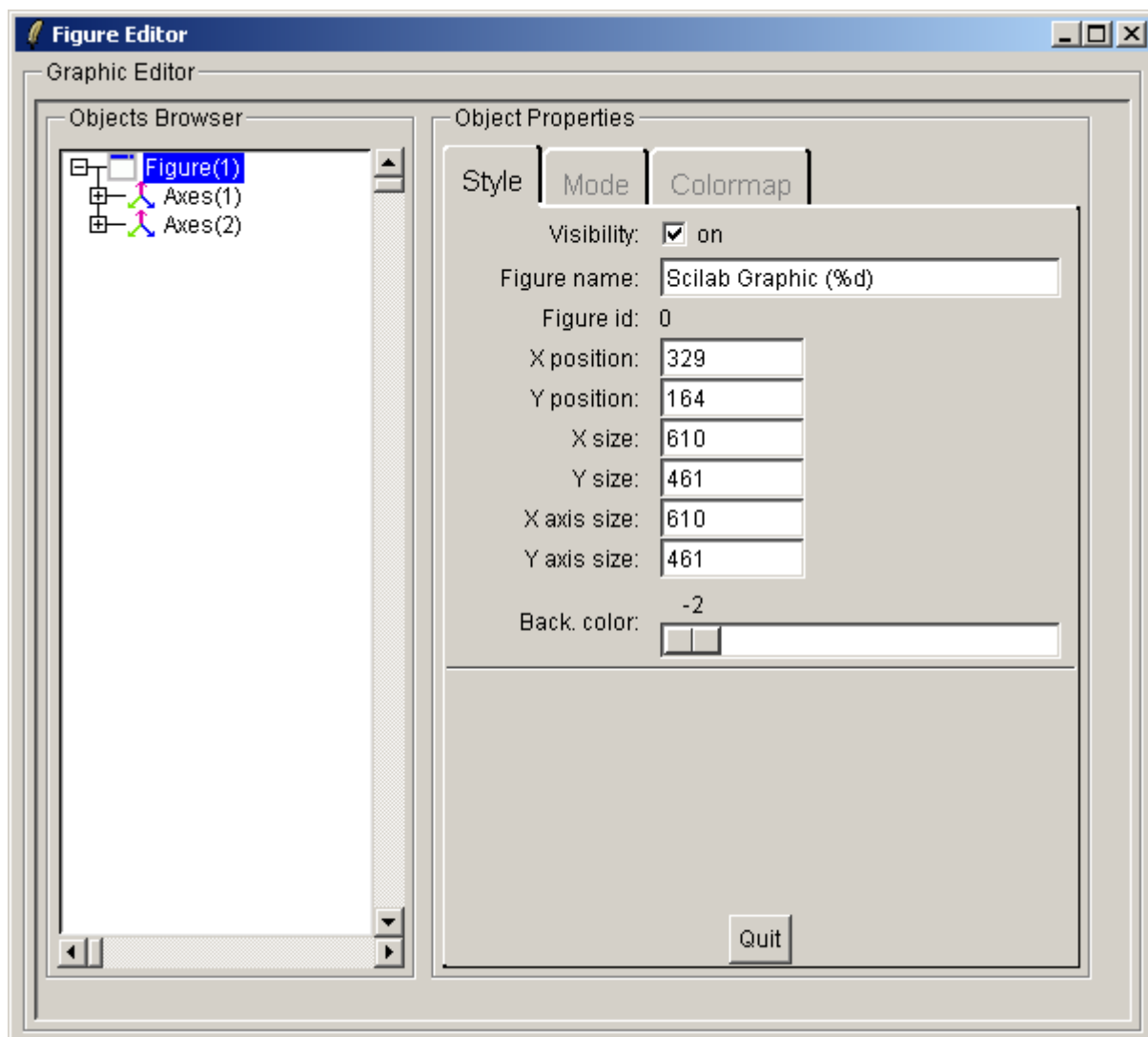


Рис. 4.20.

5.4. Построение нескольких графиков в одном графическом окне

Для построения нескольких графиков в одном графическом окне необходимо сформировать область в этом окне и в ней вывести график. Для формирования области в графическом окне служит команда

```
plotframe(rect, tics [,grid,bound, title, x-leg, y-leg, x, y, w, h])
```

`rect` – вектор $[x_{\min}, y_{\min}, x_{\max}, y_{\max}]$, который определяет границы изменения x и y координат области;

`tics` – вектор $[n_x, m_x, n_y, m_y]$, который определяет количество линий сетки по оси X (m_x) и Y (m_y), величины n_x и n_y должны определять число подинтервалов по осям X и Y ;

`grid` – логическая переменная, которая определяет наличие или отсутствие координатной сетки;

`bound` – логическая переменная, которая при значении `true` позволяет игнорировать параметры `tics(2)` и `tics(4)`;

`title` – заголовок, который будет выводиться над графическим окном;

`x-leg, y-leg` – подписи x и y осей графика;

x, y - координаты верхнего левого угла области в графическом окне, w – ширина, h – высота окна, значения x, y, w, h измеряются в относительных единицах и лежат в диапазоне $[0, 1]$.

После определения области в него можно вывести график функции с помощью `plot`, `plot2d` и т.д.

Рассмотрим пример построения четырех осей координат в графическом окне и вывода в каждую из них соответствующего графика $y=\sin(2x)$, $z=\cos(3x)$, $u=\cos(\sin(2x))$ и $v=\sin(\cos(3x))$.

```
x=[-10:0.01:10];
y=sin(2*x);
z=cos(3*x);
u=cos(sin(2*x));
v=sin(cos(3*x));
rect=[min(x), -1, max(x), 1];
tics=[2, 11, 10, 5];
plotframe(rect, tics, [%t, %t], ["Function   y=sin(2x)", "X", "Y"],
[0, 0, 0.5, 0.5])
plot(x, y);
plotframe(rect, tics, [%f, %f], ["Function   y=cos(3x)", "X", "Y"],
[0.5, 0, 0.5, 0.5])
plot(x, z);
plotframe(rect, tics, [%f, %f], ["Function y=cos(sin(2x))", "X", "Y"],
[0, 0.5, 0.5, 0.5])
plot(x, u);
plotframe(rect, tics, [%t, %t], ["Function y=sin(cos(3x))", "X", "Y"],
[0.5, 0.5, 0.5, 0.5])
plot(x, v);
```

Листинг 4.15

Получаемый график изображен на рис. 4.21.

Еще одним способом изображения нескольких графиков в одном окне является использование функции `subplot`, которая разделяет графическое окно на несколько отдельных графиков. Обращение к ней имеет вид:

```
subplot(m, n, p) или subplot(mnp)
```

Графическое окно разбивается на m окон по вертикали и n окон по горизонтали, текущим окном становится окно с номером p .

В качестве примера рассмотрим построение шести графиков $y=\sin(x)$, $z=\cos(x)$, $u=\cos(\sin(x))$ и $v=\sin(\cos(x))$, $w=\exp(\sin(x))$, $r=\exp(\cos(x))$ (см. листинг 4.16 и рис. 4.22).

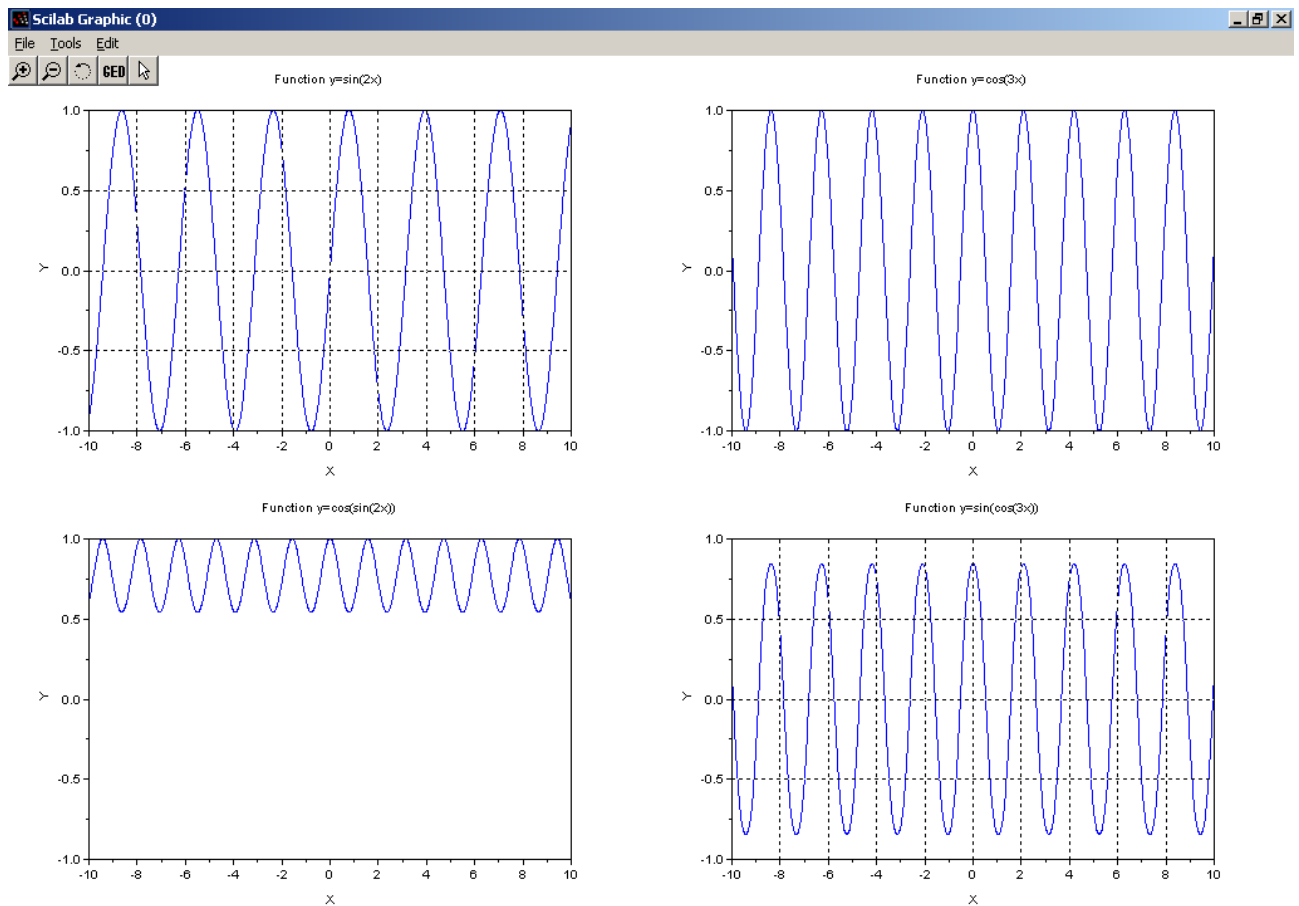


Рис. 4.21. Графики четырех функций в одном графическом окне

```
x=[-10:0.01:10];
y=sin(x);
z=cos(x);
u=cos(sin(x));
v=sin(cos(x));
w=exp(sin(x));
r=exp(cos(x));
subplot(3,2,1);
plot(x,y);
xtitle('Function y=sin(x) ','X','Y');
subplot(3,2,2);
plot(x,z);
xtitle('Function z=cos(x) ','X','Z');
subplot(3,2,3);
plot(x,u);
xtitle('Function u=cos(sin(x)) ','X','U');
subplot(3,2,4);
plot(x,v);
xtitle('Function v=sin(cos(x)) ','X','V');
subplot(3,2,5);
plot(x,w);
xtitle('Function w=exp(sin(x)) ','X','W');
subplot(3,2,6);
plot(x,r);
xtitle('Function r=sin(x) ','X','R');
```

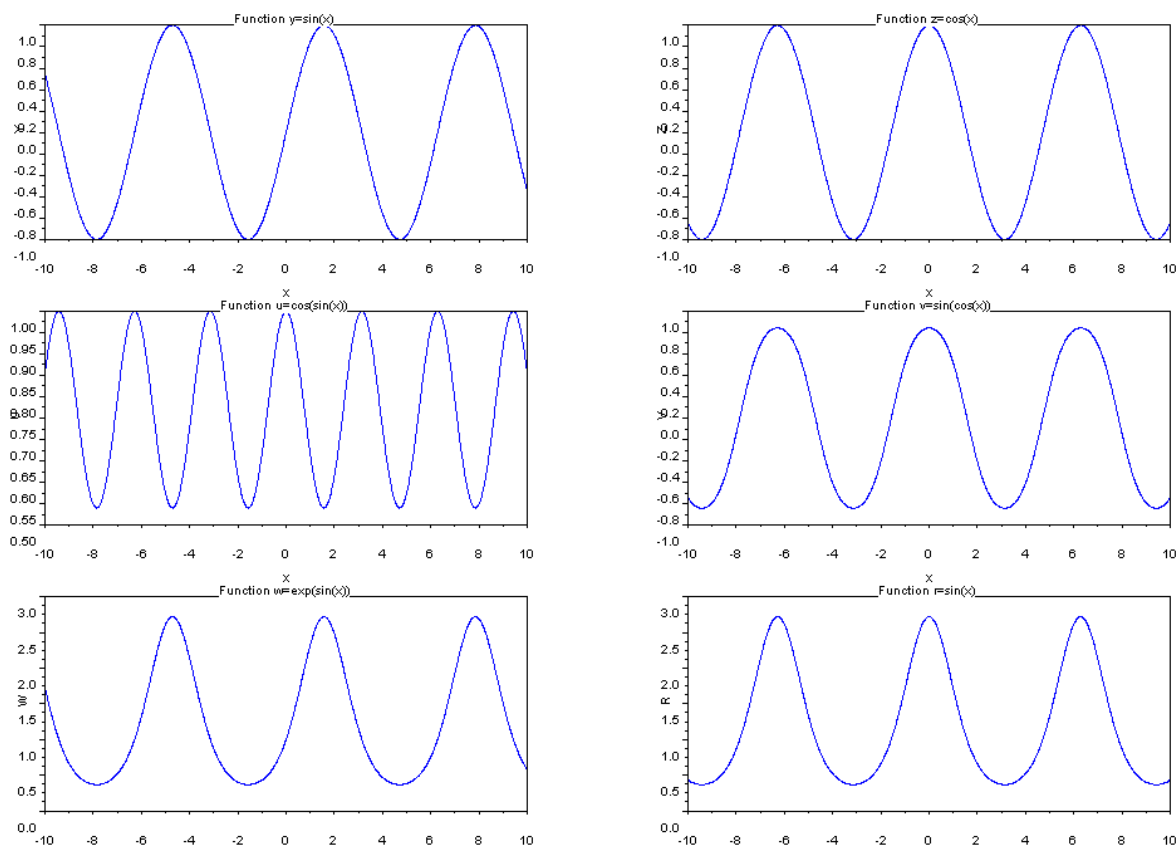
Листинг 4.16.

Рис. 4.22. Графики шести функций в одном графическом окне

Любой график можно экспортировать в графический файл для этого в окне графика выбрать команду File – Export и в появившемся окне (см. рис. 4.23) выбрать тип сохраняемого файла, цвет (Color(цветной), Black and White (черно-белый)), ориентация рисунка. После это можно выбрать папку и имя экспортируемого файла (см. рис. 4.24).

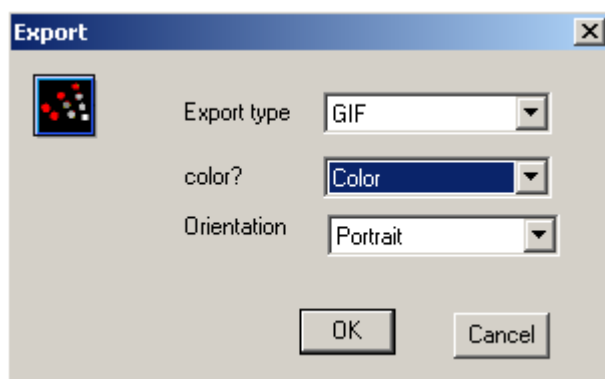


Рис. 4.23. Параметры экспорта графика в графический файл.

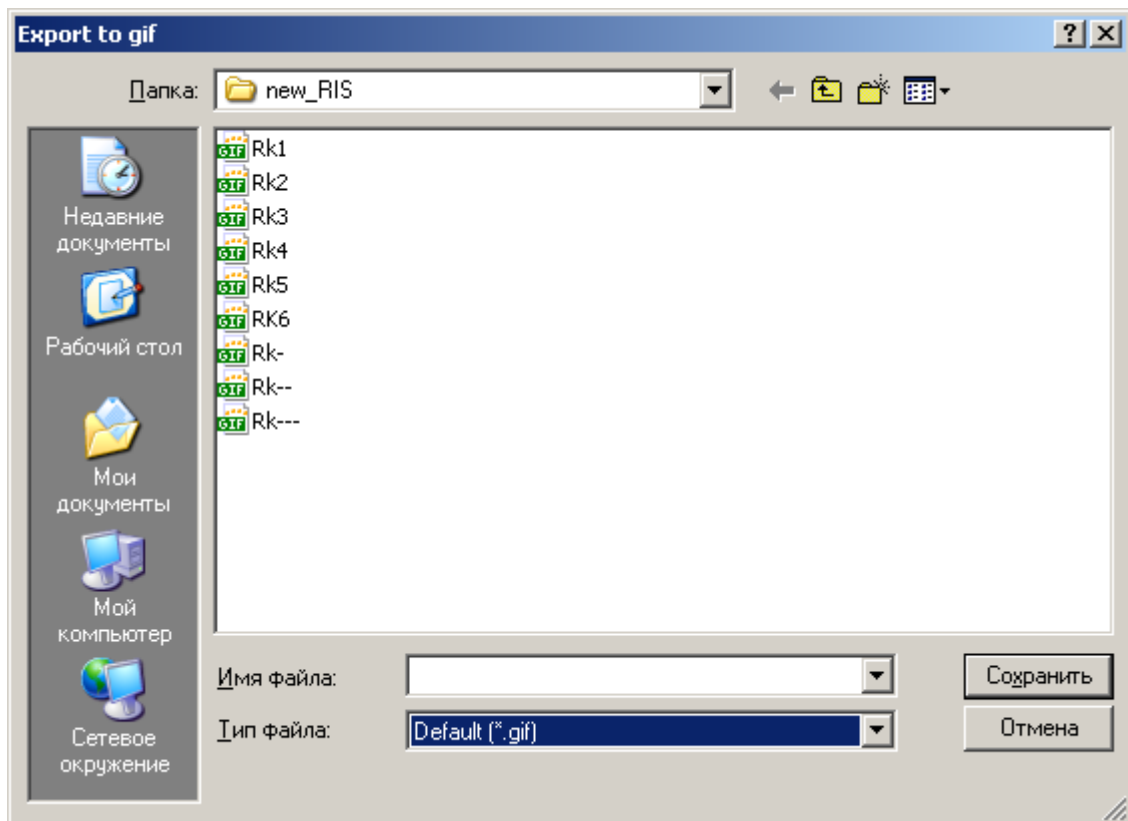


Рис. 4.24. Выбор папки и файла для экспорта графика

5. Построение трехмерных графиков в Scilab

5.1. Возможности Scilab для построения поверхностей

В Scilab 3 для построения поверхностей $z=f(x, y)$ предназначены функции

```
plot3d(x, y, z),
plot3d1(x, y, z),
```

здесь x – вектор-столбец значений абсцисс; y – вектор-столбец значений ординат; z – матрица значений функции a в узлах сетки;

Отличие функций состоит в том, что `plot3d` строит каркасный график (см. рис. 5.1), а `plot3d1` – каркасную поверхность, заливая ее каждую клетку цветом, который зависит от значения функции в узлах сетки (см. рис. 5.2).

Построение трехмерного графика рассмотрим на примере функции $z(x, y) = 5y^2 - x^2$ в области $x \in [-2; 2]$, $y \in [-3; 3]$ (см. листинг 5.1, рис. 5.1).

```
x=[-2:0.1:2];
y=[-3:0.1:3];
for i=1:length(x)
for j=1:length(y)
z(i,j)=5*y(j)^2-x(i)^2;
end
end
plot3d(x', y', z, 35, 45);
// Здесь 35 и 45 угол поворота наблюдателя
```

Листинг 5.1

Если функцию `plot3d(x', y', z, 35, 45)` заменить функцией `plot3d1(x', y', z, 35, 45)` график поверхности примет вид, изображенный на рис. 5.2.

В Scilab 4.0 строить графики поверхностей стало удобнее, возможности пакета стали похожи на построение графиков в MATLAB 7.0.

Для построения графика двух переменных $z=f(x, y)$ необходимо выполнить следующие действия.

1. Сформировать в области построения графика прямоугольную сетку, проводя прямые, параллельные осям $y=y_j$ и $x=x_i$, где

$$x_i = x_0 + ih, h = \frac{x_n - x_0}{n}, i = 0, 1, \dots, n, \quad y_j = y_0 + jh, h = \frac{y_k - y_0}{k}, j = 0, 1, \dots, k.$$

2. Вычислить значения $z_{i,j} = f(x_i, y_j)$ во всех узлах сетки.
3. Обратиться к функции построения поверхности, передавая ей в качестве параметров сетку и матрицу $Z = \{z_{i,j}\}$ значений в узлах сетки.

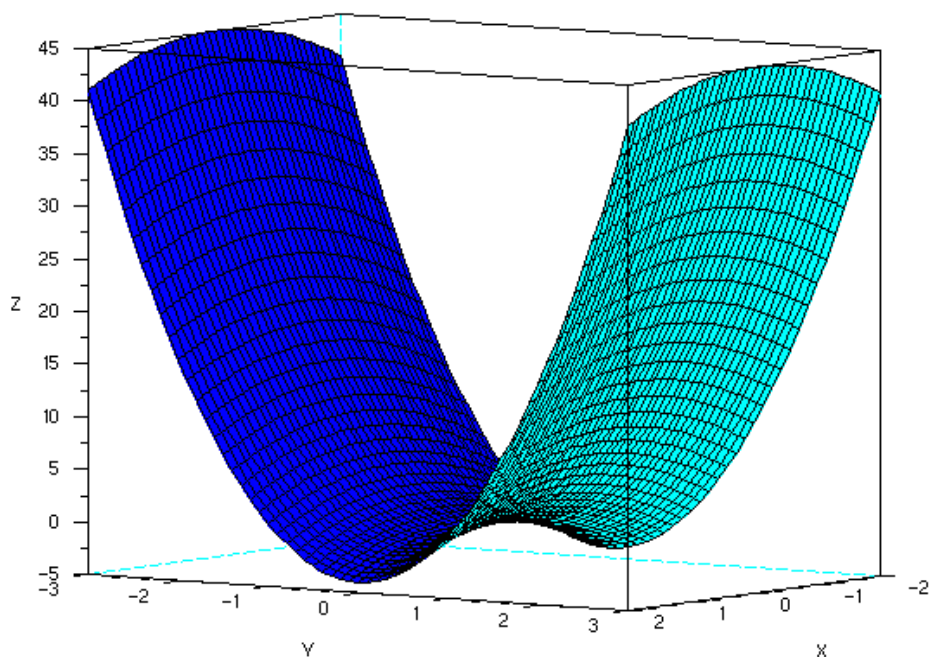


Рис. 5.1. График функции $z(x, y) = 5y^2 - x^2$, построенный с помощью `plot3d`

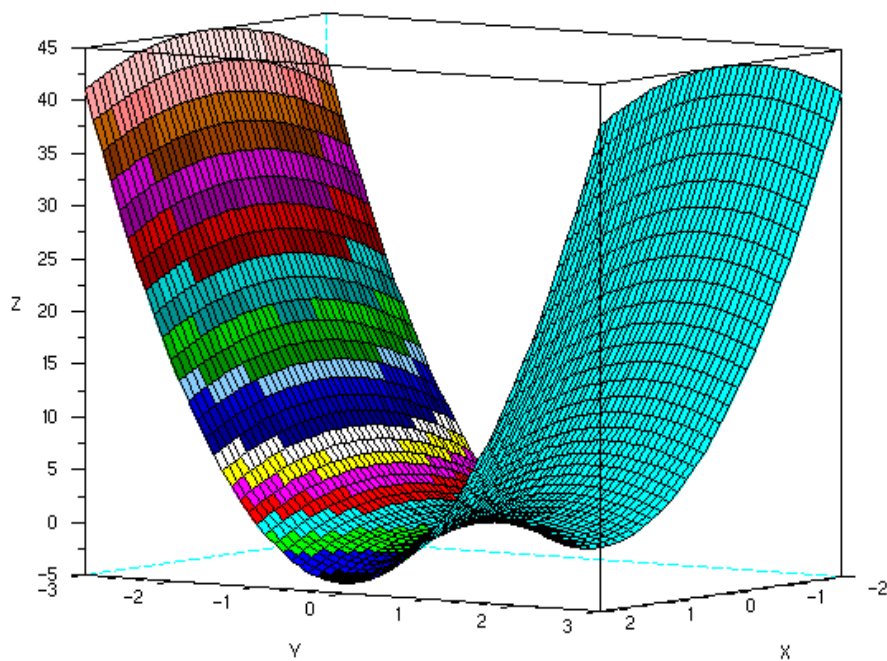


Рис. 5.2. График функции $z(x, y) = 5y^2 - x^2$, построенный с помощью `plot3d`

Для формирования прямоугольной сетки в Scilab 4.0 появилась функция `meshgrid`.

Рассмотрим построение 3-х мерного графика на следующем примере функции $z(x,y)=5y^2-x^2$ $x \in [-2; 2]$, $y \in [-3; 3]$. Для формирования сетки воспользуемся функцией `meshgrid`.

```
[x y]=meshgrid(-2:2,-3:3)
//Здесь -2:2 -массив, определяющий сетку по X,
// -3:3 - массив, определяющий сетку по Y
x =
-2 -1 0 1 2
-2 -1 0 1 2
-2 -1 0 1 2
-2 -1 0 1 2
-2 -1 0 1 2
-2 -1 0 1 2
-2 -1 0 1 2
-2 -1 0 1 2
y =
-3 -3 -3 -3 -3
-2 -2 -2 -2 -2
-1 -1 -1 -1 -1
0 0 0 0 0
1 1 1 1 1
2 2 2 2 2
3 3 3 3 3
```

После формирования сетки вычислим значение z во всех узлах

```
>> z=5*y.^2-x.^2
z =
41. 44. 45. 44. 41.
16. 19. 20. 19. 16.
1. 4. 5. 4. 1.
- 4. - 1. 0. - 1. - 4.
1. 4. 5. 4. 1.
16. 19. 20. 19. 16.
41. 44. 45. 44. 41.
```

Затем обратимся к функции `mesh` для построения графика

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

В результате чего будет построен трехмерный график (см. рис. 5.3).

Для получения менее грубого графика следует сетку делать более плотной (рис. 5.4)¹.

```
[x y]=meshgrid(-2:0.1:2,-3:0.1:3);
z=5*y.^2-x.^2;
mesh(x, y, z);
```

Листинг 5.2

Кроме построения каркасного графика с помощью функции `mesh` в Scilab 4.0 есть функция `surf`, которая строит каркасную поверхность, заливая ее каждую клетку цветом, который зависит от значения функции в узлах сетки. Использование функции `surf` рассмотрим при построении графика функции $z(x, y)=\sqrt{3x^2+2y^2}$. Решение задачи с

¹ Отличие между рис. 5.2 и 5.3-5.4 обусловлено различными углами обзора

помощью функции `surf` представлено на листинге 5.3, график изображен на рис. 5.5.

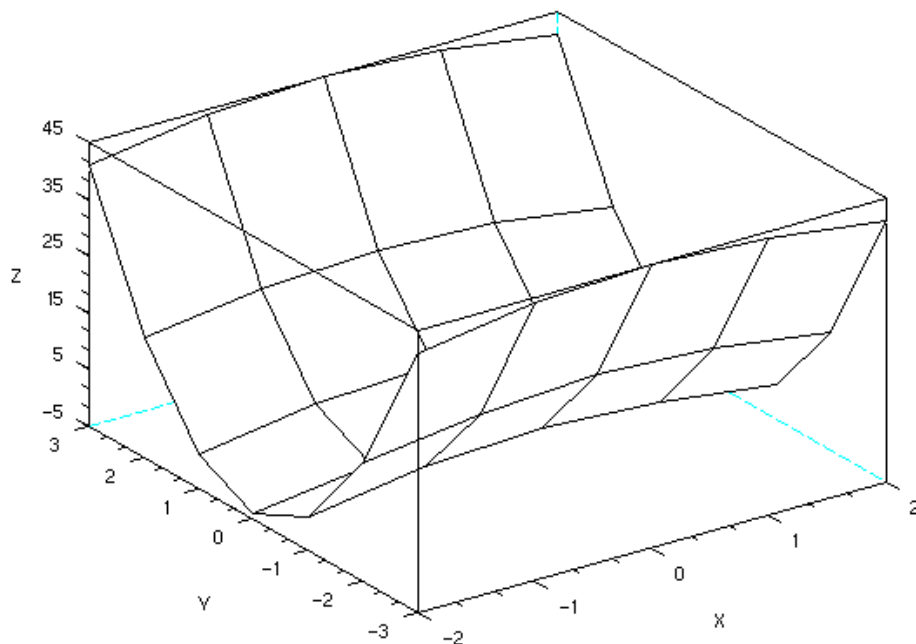


Рис. 5.3. График функции $z(x, y) = 5y^2 - x^2$, построенный с помощью функции `mesh`

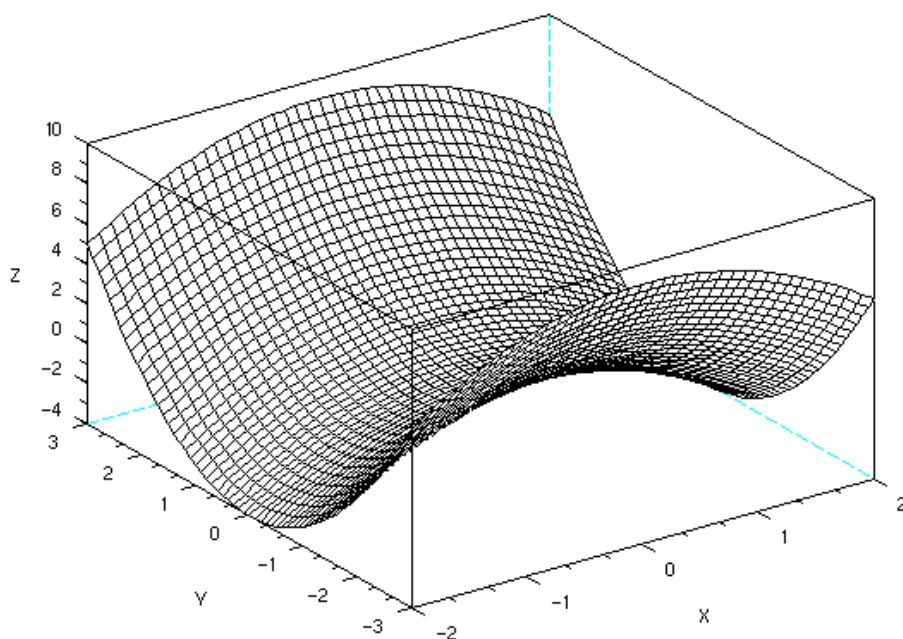


Рис. 5.4. График функции $z(x, y) = 5y^2 - x^2$ с плотной сеткой, построенный с помощью функции `mesh`

```
[x y]=meshgrid(-2:0.2:2,-2:0.2:2);
z=sqrt(x.^2+y.^2);
surf(x,y,z);
```

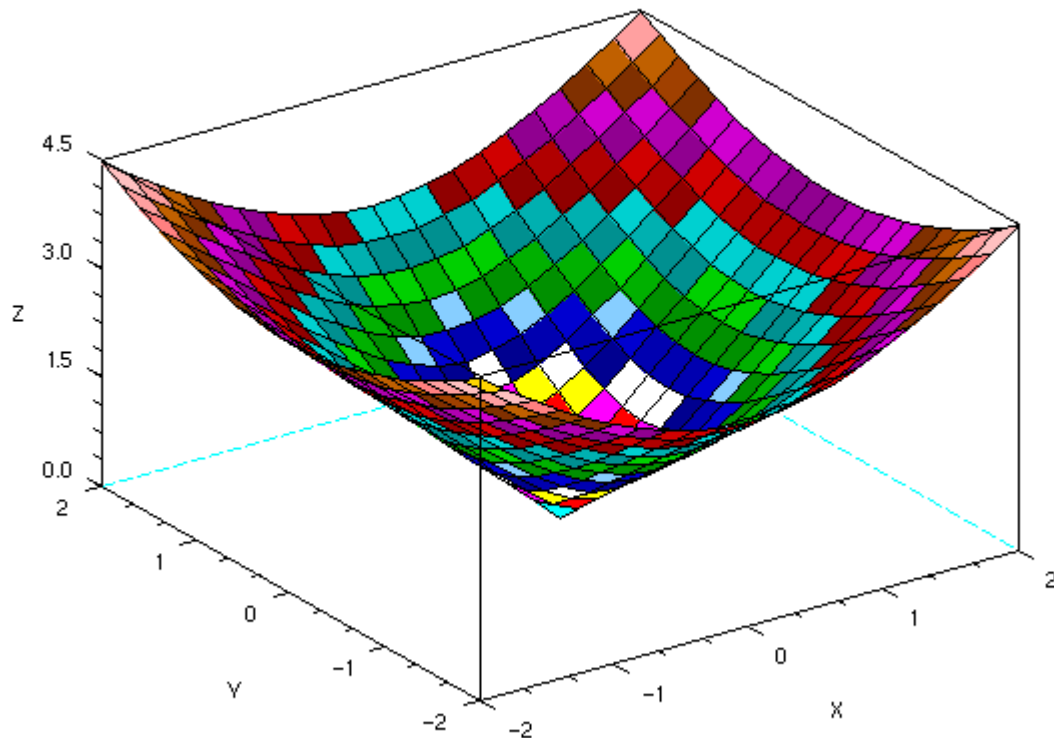
Листинг 5.3

Рис. 5.5. График функции, построенный с помощью surf

В Scilab можно построить графики двух поверхностей в одной системе координат, для этого, как и для плоских графиков следует использовать команду `mtlb_hold('on')`, которая блокирует создание второго нового окна при выполнении команд `surf` или `mesh`.

Построить график функции $z(x, y) = \pm(3x^2 + 4y^2) - 1$. Решение задачи с помощью функции `surf` представлено ниже, полученный график изображен на рис. 5.6.

```
[x y]=meshgrid(-2:0.2:2,-2:0.2:2);
z=3*x.^2+4*y.^2-1;
z1=-3*x.^2-4*y.^2-1;
surf(x,y,z);
mtlb_hold('on');
surf(x,y,z1);
```

Листинг 5.4.

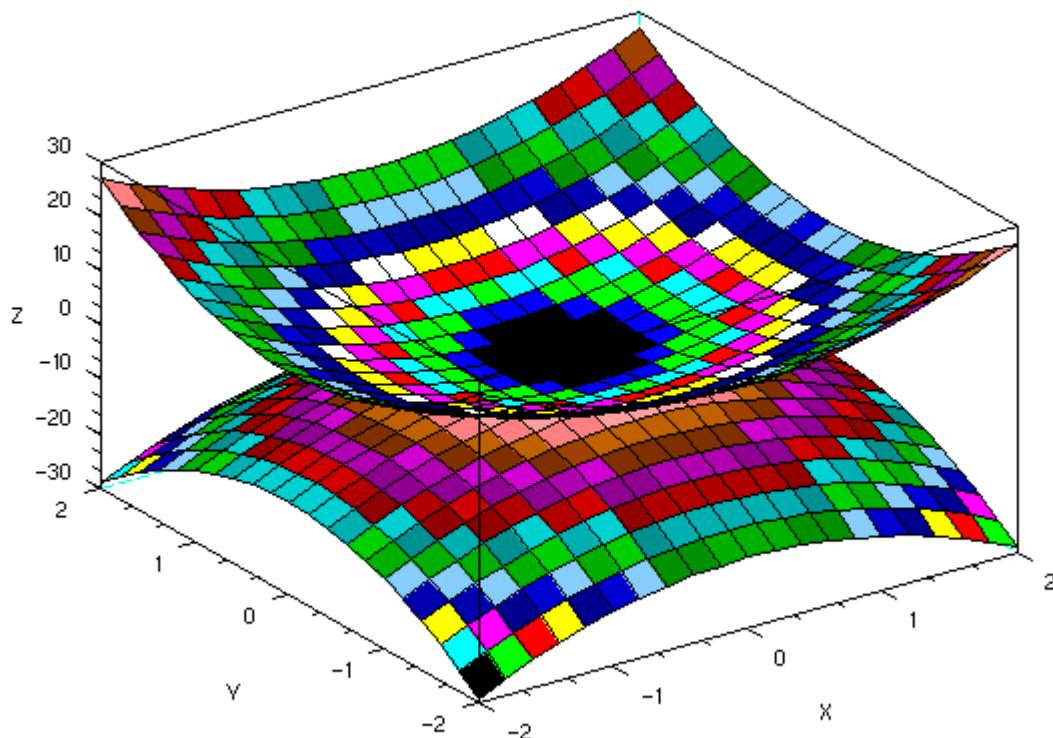


Рис. 5.6. График функции $z(x, y) = \pm(3x^2 + 4y^2) - 1$

5.2. Построение графиков поверхностей, заданных параметрически

При построении графиков поверхностей, заданных параметрически $x(u, v)$, $y(u, v)$ и $z(u, v)$ необходимо построить матрицы X , Y и Z одинакового размера. Для этого массивы u и v должны быть одинакового размера. После этого следует выделить два основных вида представления x , y и z в случае параметрического задания поверхностей:

1. Если x , y и z представимы в виде $f(u)g(v)$, то соответствующие им матрицы X , Y и Z следует формировать в виде матричного умножения $f(u)$ на $g(v)$.
2. Если x , y и z представимы в виде $f(u)$ или $g(v)$, то в этом случае матрицы X , Y и Z следует записывать в виде $f(u) \cdot \text{ones}(\text{size}(v))$ или $g(v) \cdot \text{ones}(\text{size}(u))$ соответственно.

Рассмотрим задачу построения графика поверхности сферы $x(u, v) = \cos(u) \cos(v)$, $y(u, v) = \cos(u) \sin(v)$, $z(u, v) = \sin(u)$.

```
u = linspace(-%pi/2, %pi/2, 40);
v = linspace(0, 2*%pi, 20);
```

```

X = cos(u)'*cos(v);
Y = cos(u)'*sin(v);
Z = sin(u)'*ones(v);
plot3d3(X,Y,Z);
// Подпись графика
xlabel('Function w=exp(sin(x))','X','Y','Z');

```

Листинг 5.5.

Получится график сферы (см. рис. 5.7).

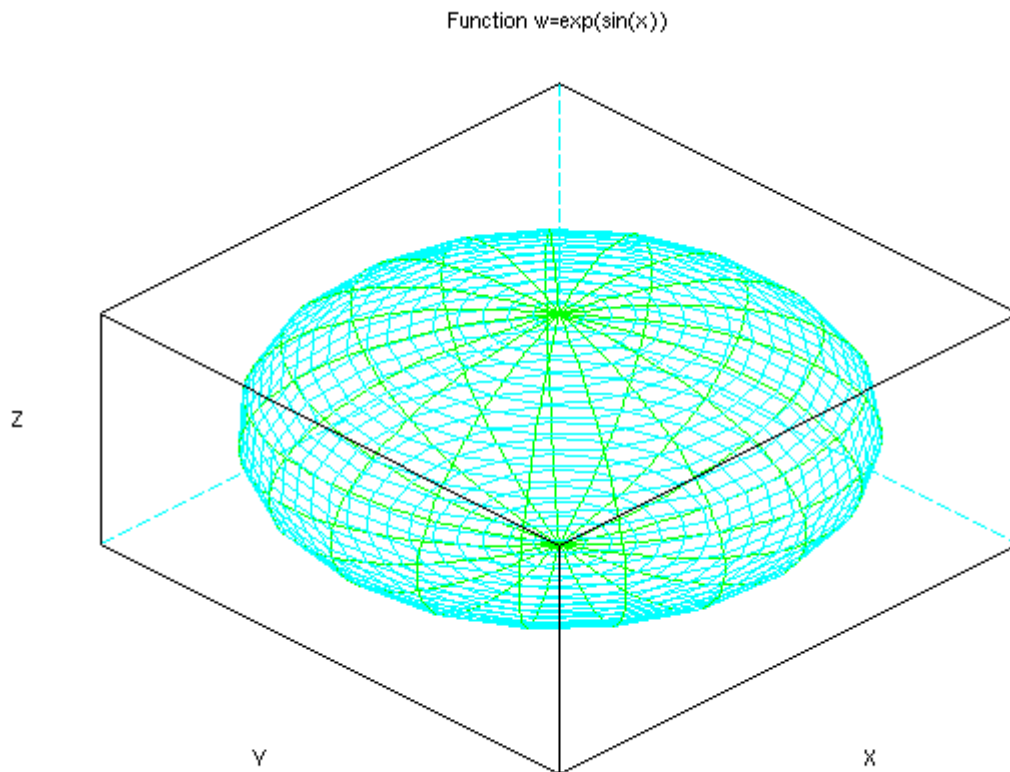


Рис. 5.7. График сферы

5.3. Построение трехмерной линии, заданной параметрически

В качестве примера рассмотрим построение трехмерной линии, заданной уравнением

$$\begin{aligned}
 x(t) &= \sin(t), \\
 y(t) &= \cos(t), \\
 z(t) &= \frac{t}{10}.
 \end{aligned}
 \quad (\text{см. листинг 5.6 и рис. 5.8})$$

```

t = 0:0.1:10*pi;
param3d(sin(t),cos(t),t/10,45,35);

```

Листинг 5.6.

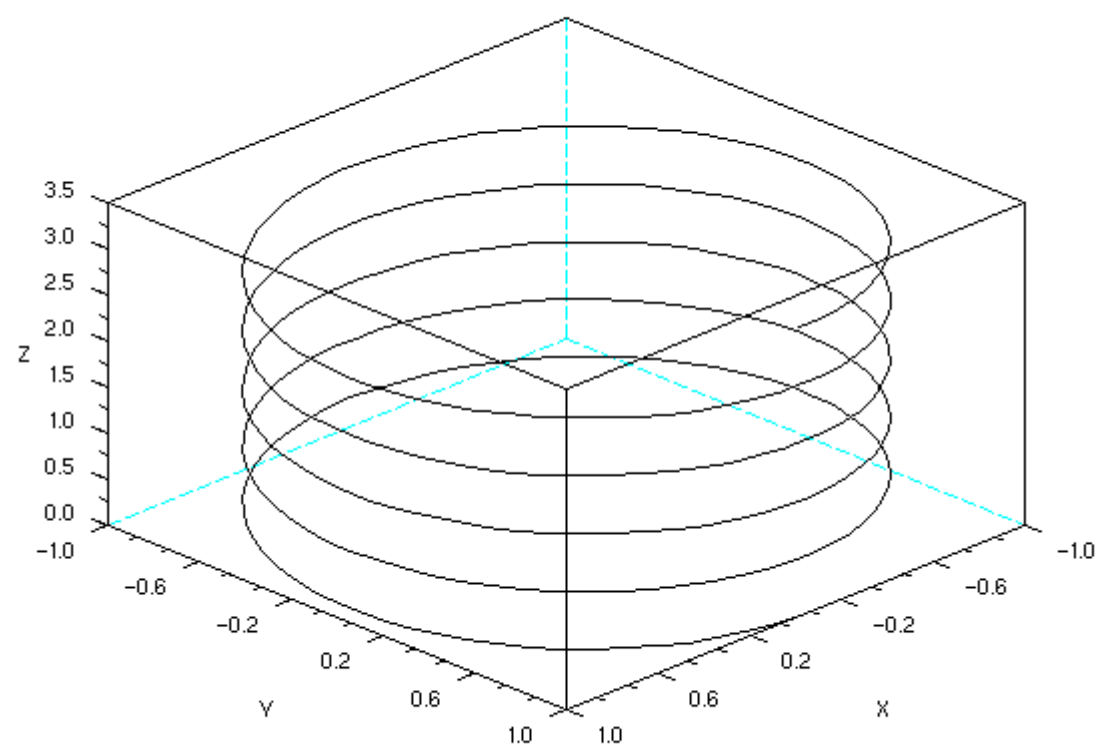


Рис.5.8. График винтовой линии

6. Нелинейные уравнения и системы в SCILAB

Если нелинейное уравнение достаточно сложное, то отыскание его корней процесс нетривиальный. Рассмотрим, какими средствами обладает Scilab для решения этой задачи.

6.1. Решение нелинейных уравнений

В общем случае аналитическое решение уравнения $f(x)=0$ можно найти только для узкого класса функций. Чаще всего приходится решать это уравнение численными методами.

Численное решение уравнения проводят в два этапа:

- *отделяют корни уравнения*, т.е. находят достаточно тесные промежутки, в которых содержится только один корень, эти промежутки называют *интервалами изоляции корня*, определить их можно, изобразив график функции или любым другим методом основанным на том, что непрерывная функции $f(x)$ имеет на интервале $[a,b]$ хотя бы один корень, если она поменяла знак $f(a) \cdot f(b) < 0$, a и b называют *пределами интервала изоляции*;
- на втором этапе проводят *уточнение отделенных корней*, т.е. находят корни с заданной точностью.

6.2.1. Алгебраические уравнения

Любое уравнение $P(x)=0$, где $P(x)$ это многочлен, отличный от нулевого, называется *алгебраическим уравнением* (полиномом относительно переменных x). Всякое алгебраическое уравнение относительно x можно записать в виде

$$a_0x^n + a_1x^{n-1} + \dots + a_{n-1}x + a_n = 0,$$

где $a_0 \neq 0$, $n \geq 1$ и a_i – коэффициенты алгебраического уравнения n -й степени. Например, *линейное уравнение* это алгебраическое уравнение первой степени, *квадратное* – второй, *кубическое* – третьей и так далее.

Для определения алгебраического уравнения в Scilab существует функция

$$\text{poly}(a, "x", ["fl"]),$$

где a это число или матрица чисел, x – символьная переменная, fl – строковая переменная, определяющая способ задания полинома. Строковая переменная fl может принимать только два значения – "roots" или "coeff" (соответственно "r" или "c"). Если $fl = c$, то будет сформирован полином с коэффициентами, хранящимися в параметре a . Если же $fl = r$, то значения параметра a воспринимаются функцией как корни, для которых необходимо рассчитать коэффициенты соответствующего полинома. По умолчанию $fl = r$.

Рассмотрим несколько примеров формирования полиномов.

Листинг 6.1 отражает создание полинома p , имеющего в качестве корня тройку, и

полинома f с коэффициентом 3.

```
-->p=poly(3, 'x', 'r');
-->f=poly(3, 'x', 'c');
-->p
p =
- 3 + x
-->f
f =
3
```

Листинг 6.1.

На листинге 6.2 приведены примеры создания более сложных полиномов.

```
-->poly([1 0 2], 'x')
ans =
      2      3
2x - 3x + x
-->poly([1 0 2], 'x', 'c')
ans =
      2
1 + 2x
-->poly([-2 2], 'x')
ans =
      2
- 4 + x
```

Листинг 6.2

Листинг 6.3 содержит примеры операций с полиномами.

```
-->p1=poly([1 -2], 'x', 'c')
p1 =
1 - 2x
-->p2=poly([3 -2], 'x', 'c')
p2 =
3 - 2x
-->p1/p2
ans =
1 - 2x
-----
3 - 2x
-->p1*p2
ans =
      2
3 - 8x + 4x
```

Листинг 6.3

Решим несколько алгебраических уравнений.

ЗАДАЧА 6.1. Найти корни полинома $2x^4 - 8x^3 + 8x^2 - 1 = 0$.

Для решения этой задачи необходимо задать полином p . Сделаем это при помощи функции `poly`, предварительно определив вектор коэффициентов V . Обратите внимание, что в уравнении отсутствует переменная x в первой степени, это означает, что соответствующий коэффициент равен нулю. Отыскание корней полинома при помощи функции `roots(p)` приведено в листинге 6.4. Графическое решение, показанное на рис. 6.1 позволяет убедиться,

что корни найдены верно.

```
-->V=[-1 0 8 -8 2];
-->p=poly(V, 'x', 'c')
p =
      2      3      4
    - 1 + 8x - 8x + 2x
-->X=roots(p)
X =

!   0.4588039 !
! - 0.3065630 !
!   1.5411961 !
!   2.306563  !
```

Листинг 6.4

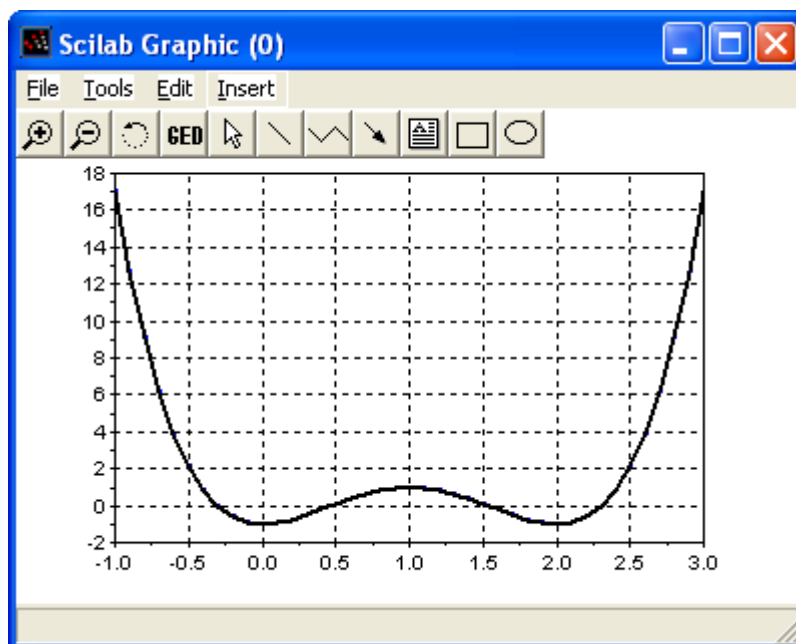


Рис. 6.1. График функции $y=2x^4-8x^3+8x^2-1$

ЗАДАЧА 6.2. Найти корни полинома $x^3+0.4x^2+0.6x-1=0$.

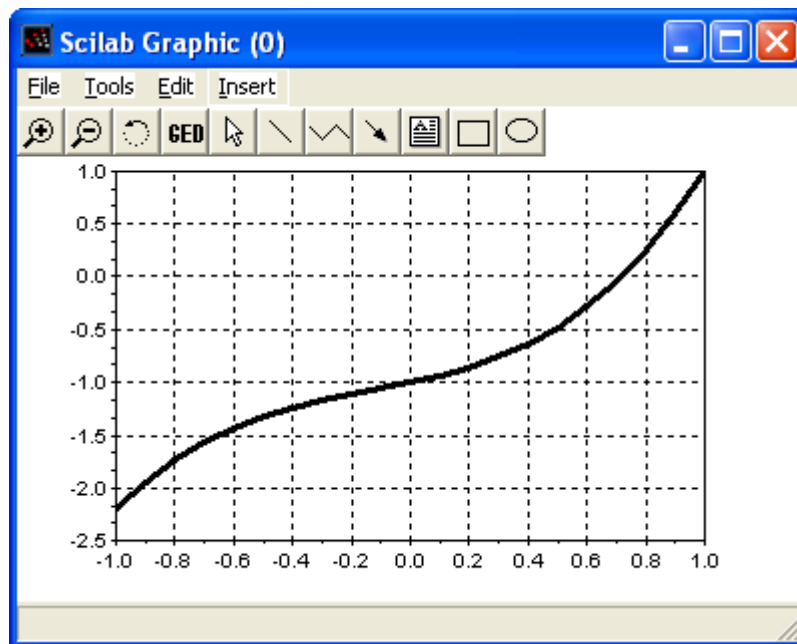
Решение этой задачи аналогично решению предыдущей. Разница заключается в способе вызова необходимых для этого функций. Не трудно заметить (листинг 6.5), что полином имеет один действительный и два комплексных корня, в отличие от полинома из задачи 6.1, в котором все корни действительные. Рис. 6.2 подтверждает это.

```
-->roots(poly([-1 0.6 0.4 1], 'x', 'c'))
ans =
!   0.7153636          !
! - 0.5576818 + 1.0425361i !
! - 0.5576818 - 1.0425361i !
```

Листинг 6.5

ЗАДАЧА 6.3. Найти решение уравнения $y(x)=0$, если $y(x)=x^4-18x^2+6$.

Листинг 6.6 демонстрирует решение этой задачи. Обратите внимание на способ определения полинома.

Рис. 6.2. График функции $y = x^3 + 0.4x^2 + 0.6x - 1$

```
-->x=poly(0, 'x');
-->y=x^4-18*x^2+.6;
-->roots(y)
ans =
!  0.1827438 !
! - 0.1827438 !
! - 4.2387032 !
!  4.2387032 !
```

Листинг 6.6

В Scilab существует функция `fsolve(x0, f)`, которую так же можно применить для решения алгебраических уравнений. Подробно эта функция будет описана далее, так как ее можно использовать для решения нелинейных уравнений, отличных от алгебраических, и для решения систем линейных и нелинейных уравнений.

ЗАДАЧА 6.4. Найти решение уравнения $y(x)=0$, если $y(x)=x^5-x^3+1$.

Решим эту задачу при помощи функции `fsolve(x0, f)`, где x_0 – начальное приближение, f – функция, описывающая левую часть уравнения $y(x)=0$. Листинг 6.7 содержит ход решения задачи. В первой строке происходит определение функции $y(x)$ в виде исходного полинома. Во второй, вызывается команда `fsolve(-2, y)`, для отыскания корней функции y . В качестве начального приближения задано число -2 , так как не трудно определить (рис. 6.3), что полином имеет единственный действительный корень, в интервале от -2 до -1 .

```
-->deff('[f]=y(x)', 'f=x^5-x^3+1')
-->X=fsolve(-2, y)
X =
- 1.2365057
```

Листинг 6.7

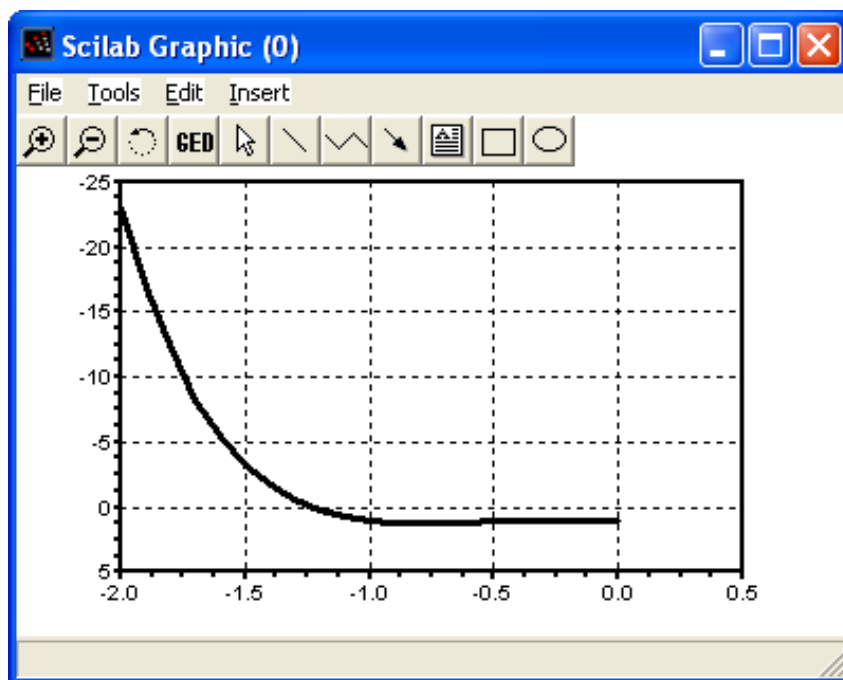


Рис. 6.3. График функции $y(x) = x^5 - x^3 + 1 = x^3 + 0.4x^2 + 0.6x - 1$

Заметим, что заданное уравнение, кроме действительного корня, имеет и мнимые. Для отыскания всех корней полинома используйте функцию `roots` (листинг 6.8).

```
-->roots(poly([1 0 0 -1 0 1], 'x', 'c'))
ans =
! 0.9590477 + 0.4283660i !
! 0.9590477 - 0.4283660i !
! -0.3407949 + 0.7854231i !
! -0.3407949 - 0.7854231i !
! -1.2365057 !
```

Листинг 6.8

Далее будет рассмотрено применение функции `fsolve` для решения неалгебраических уравнений.

6.2.2. Трансцендентные уравнения

Уравнение, в котором неизвестное входит в аргумент трансцендентных функций, называется *трансцендентным уравнением*. К трансцендентным уравнениям принадлежат показательные, логарифмические, тригонометрические.

Рассмотрим применение функции `fsolve` для решения трансцендентных уравнений.

ЗАДАЧА 6.5. Найти решение уравнения: $\sqrt[3]{(x-1)^2} - \sqrt[3]{x^2} = 0$.

Выражение, стоящее в правой части уравнения можно представить в виде разности двух функций $f(x) - g(x) = 0$, где $f(x) = \sqrt[3]{(x-1)^2}$, $g(x) = \sqrt[3]{x^2}$. Тогда решение задачи будет выглядеть, так как показано в листинге 6.9. В качестве приближенного корня был выбран ноль, т.к. на рис. 6.4 видно, абсцисса точки пересечения линий $f(x)$ и $g(x)$ лежит в интервале $[0; 1]$.

```
-->deff(' [y]=f1(x)', 'y1=((x-1)^2)^(1/3), y2=(x^2)^(1/3), y=y1-y2')
-->fsolve(0, f1)
ans =
    0.5
```

Листинг 6.9

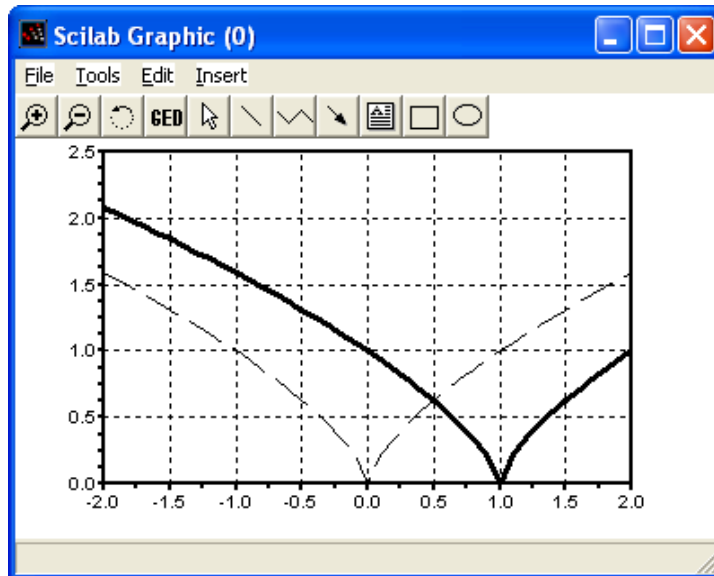
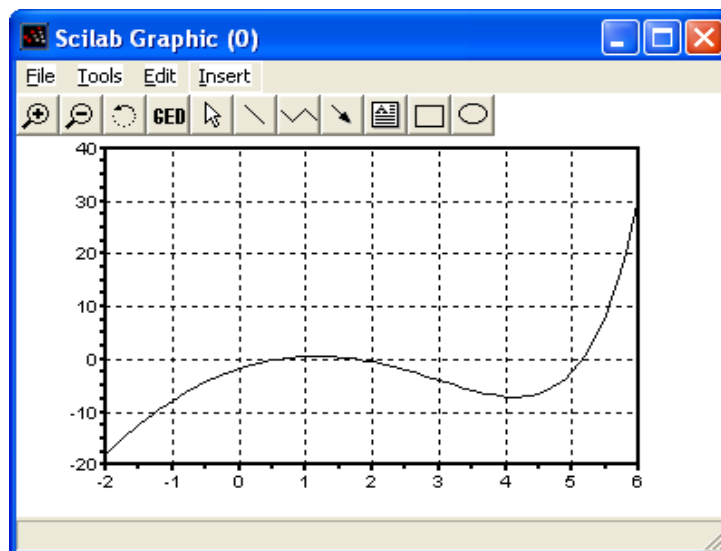
Рис. 6.4. График функций $f(x) = \sqrt[3]{(x-1)^2}$ и $g(x) = \sqrt[3]{x^2}$

Рис. 6.4.

ЗАДАЧА 6.6. Найти корни уравнения $f(x) = e^x/5 - 2(x-1)^2$.

Рис. 6.5: График функции $f(x) = e^x/5 - 2(x-1)^2$

На рис. 6.5 видно, что график функции $f(x)$ трижды пересекает ось абсцисс, то есть уравнение имеет три корня. Последовательно вызывая функцию `fsolve` с различными начальными приближениями, так как показано в листинге 6.10, получим все решения заданного уравнения.

```
-->deff(' [y]=f(x)', 'y=exp(x)/5-2*(x-1)^2')
-->x(1)=fsolve(0, f);
-->x(2)=fsolve(2, f);
-->x(3)=fsolve(5, f);
```

```
-->x
x =
! 0.5778406 !
! 1.7638701 !
! 5.1476865 !
```

Листинг 6.10

Кроме того, начальные приближения можно задать в виде вектора и тогда функцию можно вызвать один раз (листинг 6.11).

```
-->fsolve([0;2;5],f)
ans =
! 0.5778406 !
! 1.7638701 !
! 5.1476865 !
```

Листинг 6.11

ЗАДАЧА 6.7. Вычислить корни уравнения $\sin(x)-0.4x=0$ в диапазоне $[-5\pi;5\pi]$.

Решение задачи приведено в листинге 6.12.

```
-->deff(' [y]=fff(x) ', 'y=-0.4+sin(x) ')
-->V=[-5*pi:pi:5*pi];
-->X=fsolve(V,fff);
-->X
X =
!-16.11948 -12.154854 -9.8362948 -5.8716685 -3.5531095 0.4115168
2.7300758 6.6947022 9.0132611 12.977887 15.296446!
```

Листинг 6.12**6.2.3. Системы уравнений**

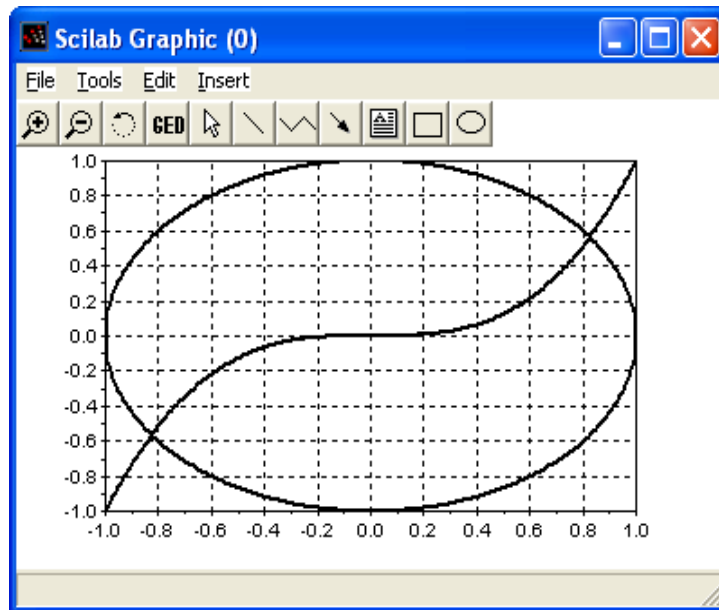
Если заданы m уравнений с n неизвестными и требуется найти последовательность из n чисел, которые одновременно удовлетворяют каждому из m уравнений, то говорят о *системе уравнений*. Для решения систем уравнений в Scilab так же применяют функцию `fsolve(x0,f)`.

ЗАДАЧА 6.8. Решить систему уравнений: $\{x^2+y^2=1; x^3-y=0\}$.

Графическое решение системы (рис. 6.6) показывает, что она имеет две пары корней. Окружность и гипербола пересекаются в точках $[0.8;0.6]$ и $[-0.8;-0.6]$. Эти значения приближительны. Для того чтобы уточнить их, применим функцию `fsolve`, предварительно определив систему с помощью файл-функции (листинг 6.13).

```
function [y]=fun(x)
y(1)=x(1)^2+x(2)^2-1;
y(2)=x(1)^3-x(2);
endfunction
-->exec('C:\Program Files\scilab-4.0-rc1\bin\fun.sce');
disp('exec done');
exec done
-->fsolve([0.5 0.5],fun)
ans =
0.8260314 0.5636242
```

```
-->fsolve([-0.5 -0.5],fun)
ans =
- 0.8260314 - 0.5636242
```

Листинг 6.13*Рис. 6.6. Графики функции $x^2+y^2=1$ и $x^3-y=0$.*

ЗАДАЧА 6.9. В данной задаче исследуется система из трех нелинейных уравнений с тремя неизвестными (листинг 6.13).

```
function [y]=fun(x)
y(1)=x(1)^2+x(2)^2+x(3)^2-1
y(2)=2*x(1)^2+x(2)^2-4*x(3)
y(3)=3*x(1)^2-4*x(2)+x(3)^2
endfunction
-->exec('D:\scilab 3\fun');disp('exec done');//вызов функции
exec done
-->fsolve([0.5 0.5 0.5],fun)//решение системы
ans =
! 0.7851969 0.4966114 0.3699228 !
```

Листинг 6.14

7. Численное интегрирование и дифференцирование

В функциях интегрирования и дифференцирования в Scilab реализованы различные численные алгоритмы.

7.1. Интегрирование по методу трапеций

В Scilab численное интегрирование по методу трапеций реализовано с помощью функции `inttrap([x,]y)`. Эта функция вычисляет площадь фигуры под графиком функции $y(x)$, которая описана набором точек (x, y) , по умолчанию элементы вектора x принимают значения номеров элементов вектора y .

ЗАДАЧА 7.1. Вычислить определенный интеграл:

$$\int_5^{13} \sqrt{2x-1} dx$$

Этот интеграл легко сводится к табличному:

$$\int_5^{13} \sqrt{2x-1} dx = \frac{\sqrt{(2x-1)^3}}{3}$$

поэтому вычислить его по формуле Ньютона–Лейбница не составит труда (листинг 7.1).

```
-->a=5;
-->b=13;
-->I=1/3*(2*b-1)^(3/2)-1/3*(2*a-1)^(3/2)
I =
32.666667
```

Листинг 7.1

Теперь применим для отыскания заданного определенного интеграла *метод трапеций*.

Листинг 7.2 содержит несколько вариантов решения данной задачи. В первом случае интервал интегрирования делится на отрезки с шагом один, во втором 0.5 и в третьем 0.1. Не трудно заметить, что чем больше точек разбиения, тем точнее значение искомого интеграла.

```
-->x=a:b;
-->y=sqrt(2*x-1);
-->inttrap(x,y)
ans =
32.655571
>>%-----
-->h=0.5;
-->x=a:h:b;
-->y=sqrt(2*x-1);
-->inttrap(x,y)
ans =
32.66389
>>%-----
-->h=0.1;
-->x=a:h:b;
-->y=sqrt(2*x-1);
```

```
-->inttrap(x,y)
ans =
    32.666556
```

Листинг 7.2

В листинге 7.3 приведен пример использования функции `inttrap` с одним аргументом. Как видим, в первом случае значение интеграла вычисленного при помощи этой функции не точно и совпадает со значением, полученным функцией `inttrap(x,y)` на интервале $[5; 13]$ с шагом 1 (листинг 7.2). То есть мы нашли сумму площадей восьми прямолинейных трапеций с основанием $h=1$ и боковыми сторонами, заданными вектором y . Во втором случае, при попытке увеличить точность интегрирования, значение интеграла существенно увеличивается. Дело в том что, уменьшив шаг разбиения интервала интегрирования до 0.1, мы увеличили количество элементов векторов x и y и применение функции `inttrap(y)` приведет к вычислению суммы площадей восьмидесяти трапеций с основанием $h=1$ и боковыми сторонами, заданными вектором y . Таким образом, в первом и втором примерах листинга 7.3 вычисляются площади совершенно разных фигур.

```
-->x=a:b;
-->y=sqrt(2*x-1);
-->inttrap(y)
ans =
    32.655571
>>%-----
-->h=0.1;
-->x=a:h:b;
-->y=sqrt(2*x-1);
-->inttrap(y)
ans =
    326.666556
```

Листинг 7.3

7.2. Интегрирование по квадратуре

Методы *трапеций* являются частными случаями *квадратурных формул Ньютона–Котеса*, которые, вообще говоря, имеют вид

$$\int_a^b y dy = (b-a) \sum_{i=0}^n H_i y_i \quad (7.1)$$

где H_i – это некоторые константы называемые постоянными Ньютона–Котеса.

Если для (7.1) принять $n=1$, то получим *метод трапеций*, а при $n=2$ – *метод Симпсона*. Эти методы называют квадратурными методами низших порядков. Для $n>2$ получают квадратурные формулы Ньютона–Котеса высших порядков.

Вычислительный алгоритм квадратурных формул реализован в Scilab функцией

```
integrate(fun, x, a, b, [,er1 [,er2]]),
```

где:

- fun – функция, задающая подынтегральное выражение в символьном виде;
- x – переменная интегрирования, так же задается в виде символа;
- a, b – пределы интегрирования (действительные числа);
- er1 и er2 – абсолютная и относительная точность вычислений (действительные числа).

ЗАДАЧА 7.2. Вычислить интеграл из задачи 7.1.

Решение приведено в листинге 7.4

```
-->integrate(' (2*x-1)^0.5', 'x', 5, 13)
ans =
    32.666667
```

Листинг 7.4

7.3. Интегрирование внешней функции

Наиболее универсальной командой интегрирования в Scilab является:

```
[I,err]=intg(a, b, name [,er1 [,er2]]),
```

где

- name – имя функции, задающей подынтегральное выражение; здесь функция может быть задана в виде набора дискретных точек (как таблица) или с помощью внешней функции;
- a, b – пределы интегрирования;
- er1 и er2 – абсолютная и относительная точность вычислений (действительные числа).

ЗАДАЧА 7.3. Вычислить интеграл из задачи 7.1

```
-->deff('y=G(x)', 'y=sqrt(2*x-1)')
-->intg(5, 13, G)
ans =
    32.666667
```

Листинг 7.5

ЗАДАЧА 7.4. Вычислить интеграл

$$\int_0^1 \frac{t^2}{\sqrt{3+\sin(t)}} dt$$

```
-->function y=f(t), y=t^2/sqrt(3+sin(t)), endfunction;
-->[I,er]=intg(0, 1, f)
er =
    1.933D-15
I =
    0.1741192
```

Листинг 7.6

7.4. Приближенное дифференцирование, основанное на интерполяционной формуле Ньютона

Идея численного дифференцирования заключается в том, что функцию $y(x)$, заданную в равноотстоящих точках x_i ($i=0, 1, \dots, n$) отрезка $[a, b]$ с помощью значений $y_i=f(x_i)$ приближенно заменяют интерполяционным полиномом Ньютона (7.2, 7.3), построенном для системы узлов x_0, x_1, \dots, x_k ($k \leq n$), и вычисляют производные $y'=f'(x)$, $y''=f''(x)$ и т.д.

$$y'(x_0) = \frac{1}{h} \left(\Delta y_0 - \frac{\Delta^2 y_0}{2} + \frac{\Delta^3 y_0}{3} - \frac{\Delta^4 y_0}{4} + \frac{\Delta^5 y_0}{5} - \dots \right), \quad (7.2)$$

На практике приближенное дифференцирование применяют в основном для функций заданных в виде таблицы.

В Scilab численное дифференцирование реализовано командой `dy=diff(y [, n])`, где y – значения функции $y(x)$ в виде вектора вещественных чисел, n – порядок дифференцирования. Результат работы функции – вектор вещественных чисел dy , представляющий собой разности порядка n интерполяционного полином Ньютона $\Delta y, \Delta^2 y, \dots, \Delta^k y$. Рассмотрим работу функции на примере.

ЗАДАЧА 7.4. Найти $y'(50)$ функции $y=\lg(x)$, заданной в виде таблицы. Решение задачи показано в листинге 7.7.

```
-->h=5;
-->x=50:5:65
x =
    50.    55.    60.    65.
-->y=log10(x)
y =
    1.69897    1.7403627    1.7781513    1.8129134
-->dy=diff(y)
dy =
    0.0413927    0.0377886    0.0347621
-->dy2=diff(y,2)
dy2 =
    - 0.0036041    - 0.0030265
-->dy3=diff(y,3)
dy3 =
    0.0005777
-->//Приближенное значение y'(50) по формуле (7.2)
-->Y=(dy(1)-dy2(1)/2+dy3(1)/3)/h
Y =
    0.0086775
-->//Значение y'(50) для lg'(x)=1/ln(10)/x
-->1/log(10)/x(1)
ans =
    0.0086859

-->//Приближенное значение y'(x), x=50,55,60 (7.2)
```

```
-->Y=(dy-dy2(1:$-1)/2+dy3(1:$-2)/3)/h
Y =
    0.0086389    0.0079181    0.0073128
-->//Значение y'(x), x=50,55,60, для lg'(x)=1/ln(10)/x
-->(1/log(10))./x(1:$-1)
ans =
    0.0086859    0.0078963    0.0072382
```

Листинг 7.7**7.5. Вычисление производной функции в точке. Приближенное вычисление частных производных.**

Более универсальной командой дифференцирования является команда

```
g=numdiff(fun,x)
```

Здесь `fun` - имя функции, задающей выражение для дифференцирования. Функция должна быть задана в виде `y=fun(x [,p1,p2,..pn])`, где `x` - переменная по которой будет проводится дифференцирование. Если параметры `p1,p2,..pn` присутствуют в описании функции, то должны быть обязательно определены при вызове, например так:

```
g=numdiff(list(fun,p1,p2,..pn),x).
```

Результат работы функции - матрица $g_{ij} = \frac{df_i}{dx_j}$.

Рассмотрим несколько примеров.

ЗАДАЧА 7.5. Вычислить $f'(1)$, если $f(x)=(x+2)^3+5x$. Решение показано в листинге 7.8.

```
-->function f=my(x), f=(x+2)^3+5*x, endfunction;
-->numdiff(my,1)
ans =
32.
-->x=1;3*(x+2)^2+5
ans =
32.
```

Листинг 7.8

ЗАДАЧА 7.6. Вычислить $f'(x)$ в точках 0, 1, 2, 3 для $f(x)=(x+2)^3+5x$ (листинг 7.9).

```
-->v=0:3;
-->numdiff(my,v)
ans =
    17.    0.    0.    0.
     0.    32.    0.    0.
     0.    0.   52.999999  0.
     0.    0.    0.   80.000002
-->function f1=my1(x), f1=3*(x+2)^2+5, endfunction;
-->my1(v)
ans =
    17.    32.    53.    80.
```

Листинг 7.9

ЗАДАЧА 7.7. Задана функция многих переменных $y(x_1, x_2, x_3) = x_1 x_2^{x_3} + x_1^2 x_3$. Вычислить

$\frac{dy}{dx_1}, \frac{dy}{dx_2}, \frac{dy}{dx_3}$ в точке (1, 2, 3). Листинг 7.10 содержит решение задачи.

```
-->function [Y]=f(X), Y=X(1)*X(2)^X(3)+X(1)^2*X(3),endfunction
-->X=[1 2 3];
-->numdiff(f,X)
ans =
    14.    12.    6.5451775
-->//-----
-->function [Y]=f1(X),
Y(1)=X(2)^X(3)+2*X(1)*X(3),
Y(2)=X(1)*X(3)*X(2)^(X(3)-1),
Y(3)=x(1)*X(2)^X(3)*log(X(2))+X(1)^2,
endfunction
-->f1(X)
ans =
    14.
    12.
    6.5451774
```

ЛИСТИНГ 7.10

8. Решение обыкновенных дифференциальных уравнений

Дифференциальным уравнением n -го порядка называется соотношение вида

$$H(t, x, x', x'', \dots, x^{(n)}) = 0 \quad (8.1)$$

Решением дифференциального уравнения является функция $x(t)$, которая обращает уравнение в тождество.

Системой дифференциальных уравнений n -го порядка называется система вида:

$$\begin{aligned} x_1' &= f_1(t, x_1, x_2, \dots, x_n) \\ x_2' &= f_2(t, x_1, x_2, \dots, x_n) \\ &\dots \\ x_n' &= f_n(t, x_1, x_2, \dots, x_n) \end{aligned} \quad (8.2)$$

Решение системы - вектор который обращает уравнения системы (8.2) в тождества:

$$x(t) = \begin{pmatrix} x_1(t) \\ x_2(t) \\ \dots \\ x_n(t) \end{pmatrix} \quad (8.3).$$

Дифференциальные уравнения и системы имеют бесконечное множество решений, которые отличаются друг от друга константами. Для однозначного определения решения требуется задать дополнительные начальные или граничные условия. Количество таких условий должно совпадать с порядком дифференциального уравнения или системы. В зависимости от вида дополнительных условий в дифференциальных уравнениях различают: *задачу Коши* – все дополнительные условия заданы в одной (чаще начальной) точке интервала; *краевую задачу* – дополнительные условия указаны на границах интервала.

Большое количество уравнений может быть решено точно. Однако есть уравнения, а особенно системы уравнений, для которых точное решение записать нельзя. Такие уравнения и системы решают при помощи численных методов. Так же численные методы применяют, если для уравнений с известным аналитическим решением требуется найти числовое значение при определенных исходных данных.

Для решения дифференциальных уравнений и систем в Sciab предусмотрена функция:

```
[y, w, iw]=ode([type], y0, t0, t [, rtol [, atol]], f [, jac] [, w, iw])
```

для которой, *обязательными входными параметрами* являются:

y_0 – вектор начальных условий;

t_0 – начальная точка интервала интегрирования;

t – координаты узлов сетки, в которых происходит поиск решения;

f – внешняя функция, определяющая правую часть уравнения или системы уравнений (8.2);

y – вектор решений (8.3).

Таким образом, для того чтобы решить обыкновенное дифференциальное уравнение вида

$$\frac{dy}{dt} = f(t, y), y(t_0) = y_0, \text{ необходимо вызвать функцию } y = \text{ode}(y_0, t_0, t, f).$$

Рассмотрим необязательные параметры функции `ode`:

`type` – параметр с помощью которого можно выбрать метод решения или тип решаемой задачи, указав одну из строк:

`"adams"` - применяют при решении дифференциальных уравнений или систем методом прогноза-коррекции Адамса;

`"stiff"` - указывают при решении жестких задач;

`"rk"` - используют при решении дифференциальных уравнений или систем методом Рунге_Кутта четвертого порядка;

`"rkf"` - указывают при выборе пятиэтапного метода Рунге_Кутта четвертого порядка;

`"fix"` - тот же метод Рунге_Кутта, но с фиксированным шагом;

`rtol, atol` - относительная и абсолютная погрешности вычислений, вектор, размерность которого совпадает с размерностью вектора `y`, по умолчанию `rtol=0.00001`, `atol=0.0000001`, при использовании параметров `"rkf"` и `"fix"` – `rtol=0.001`, `atol=0.0001`;

`jac` – матрица, представляющая собой якобиан правой части жесткой системы дифференциальных уравнений, задают матрицу в виде внешней функции вида `J=jak(t, y)`;

`w, iw` – векторы, предназначенные для сохранения информации о параметрах интегрирования, которые применяют для того, чтобы последующие вычисления выполнялись с теми же параметрами.

Рассмотрим использование функции на примере следующих задач.

ЗАДАЧА 8.1. Решить задачу Коши

$$\frac{dx}{dt} + x = \sin(xt), x(0) = 1.5.$$

Перепишем уравнение следующим образом:

$$\frac{dx}{dt} = -x + \sin(xt), x(0) = 1.5.$$

Далее представим его в виде внешней функции, так как показано в листинге 8.1 и применим функцию `y=ode(x0, t0, t, f)`, в качестве параметров которой будем использовать

- `f` – ссылка на предварительно созданную функцию `f(t, x)`;
- `t` – координаты сетки;

- x_0, t_0 – начальное условие $x(0)=1.5$;
- y – результат работы функции.

График, моделирующий процесс, описанный заданным уравнением, представлен на рис.8.1.

```
-->function yd=f(t,x),yd=-x+sin(t*x),endfunction;
-->x0=1.5;t0=0;t=0:1:35;
-->y=ode(x0,t0,t,f);
-->plot(t,y)
```

Листинг 8.1

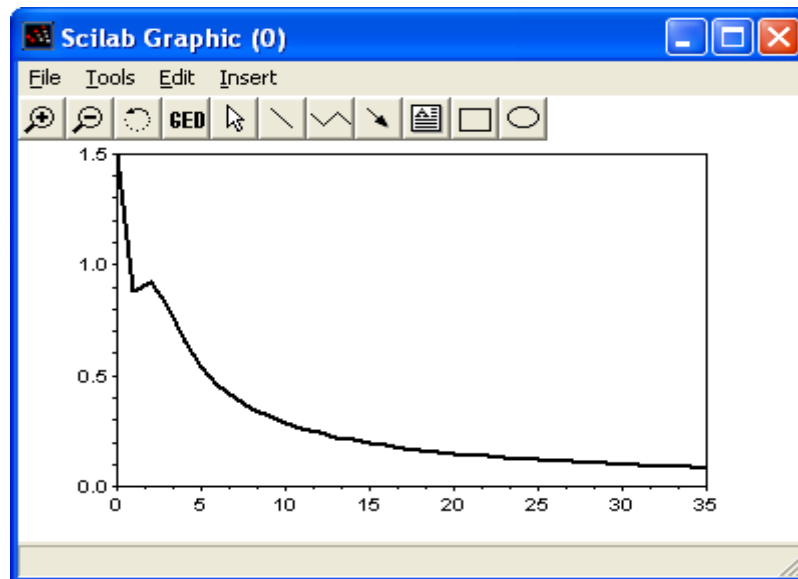


Рис. 8.1. Решение дифференциального уравнения из задачи 8.1

ЗАДАЧА 8.2. Решить задачу Коши

$$\begin{aligned}x' &= \cos(x y), \\y' &= \sin(x + t y), \\x(0) &= 0, y(0) = 0.\end{aligned}$$

на интервале $[0; 10]$.

Листинг 8.2 содержит функцию, описывающую заданную систему обыкновенных дифференциальных уравнений и команды Scilab необходимые для ее численного и графического решения (рис.8.2).

```
>>%Функция, описывающая систему дифференциальных уравнений
-->function dy=syst(t,y)
-->dy=zeros(2,1);
-->dy(1)=cos(y(1)*y(2));
-->dy(2)=sin(y(1)+y(2)*t);
-->endfunction
>>%Решение системы дифференциальных уравнений
-->x0=[0;0];t0=0;t=0:1:10;
-->y=ode(x0,t0,t,syst);
>>%Формирование графического решения
-->plot(t,y)
```

Листинг 8.2

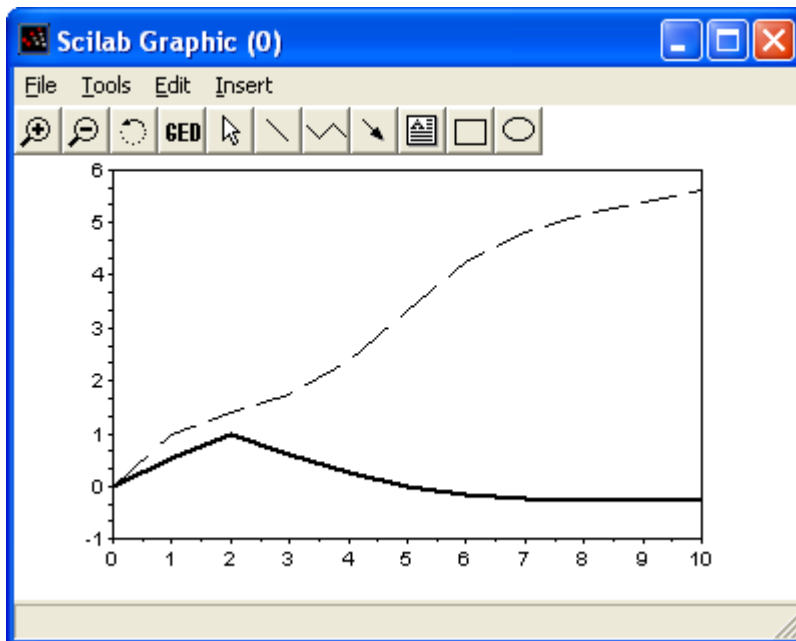


Рис. 8.2. Решение задачи 8.2

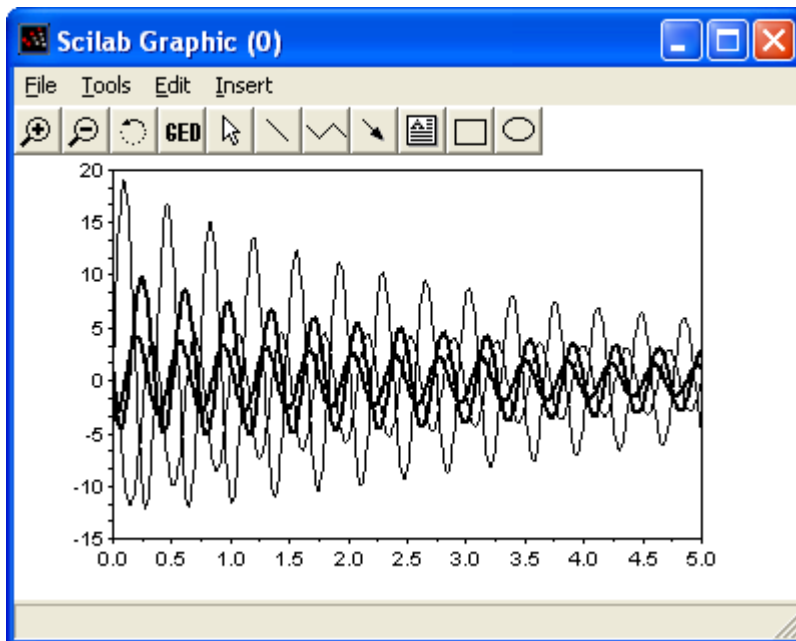


Рис. 8.3. Графическое решение жесткой системы

ЗАДАЧА 8.3. Найти решение задачи Коши для следующей жесткой системы:

$$\frac{dX}{dt} = \begin{pmatrix} 119.46 & 185.38 & 126.88 & 121.03 \\ -10.395 & -10.136 & -3.636 & 8.577 \\ -53.302 & -85.932 & -63.182 & 54.211 \\ -115.58 & -181.75 & -112.8 & -199 \end{pmatrix} X; X(0) = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} .$$

Решение системы показано в листинге 8.3. Графическое решение показано на рис. 8.3.

```
-->B=[119.46 185.38 126.88 121.03;-10.395 -10.136 -3.636 8.577;
-->-53.302 -85.932 -63.182 -54.211;-115.58 -181.75 -112.8 -199];
-->function dx=syst1(t,x), dx=B*x,endfunction
-->function J=Jac(t,y), J=B,endfunction
-->x0=[1;1;1;1]; t0=0; t=0:0.01:5;
```

```
-->y=ode("stiff",x0,t0,t,syst1,Jacobian);
-->plot(t,y)
```

Листинг 8.3

ЗАДАЧА 8.4. Решить нелинейную жесткую систему дифференциальных уравнений:

$$\left\{ \begin{array}{l} \frac{dx_1}{dt} = -7x_1 + 7x_2 \\ \frac{dx_2}{dt} = 157x_1 - 1.15x_2x_3 \\ \frac{dx_3}{dt} = 0.96x_1x_2 - 8.36x_3 \end{array} \right\}, \quad X(0) = \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix}$$

На рис. 8.4 показано решение системы на интервале [0; 2]. Команды Scilab необходимые для достижения этого решения приведены в листинге 8.4.

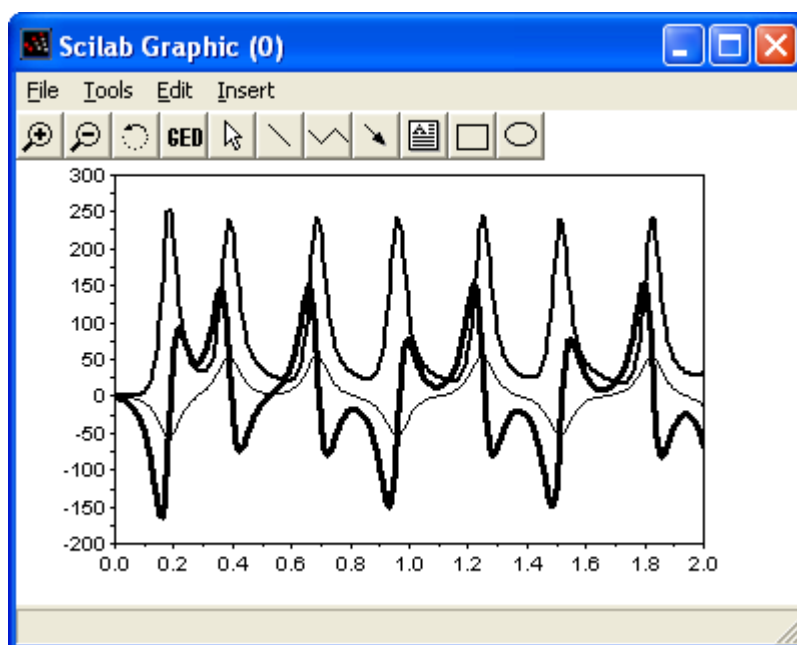


Рис. 8.4. Решение жесткой нелинейной системы

```
//Функция задающая систему ОДУ
function dx=syst2(t,x)
dx=zeros(3,1);
dx(1)=-7*x(1)+7*x(2);
dx(2)=157*x(1)+x(2)-1.15*x(1)*x(3);
dx(3)=0.96*x(1)*x(2)-8.36*x(3);
endfunction
-->>//Решение ОДУ
-->x0=[-1;0;1]; t0=0; t=0:0.01:2;y=ode("stiff",x0,t0,t,syst2);
-->plot(t,y)
```

Листинг 8.4

ЗАДАЧА 8.5. Решить следующую краевую задачу

$$\frac{d^2x}{dt^2} + 4 \frac{dx}{dt} + 13 = e^{\sin(t)}, \quad x(0.25) = -1, \quad x'(0.25) = 1.$$

на интервале [0.25; 2].

Преобразуем уравнение в систему, сделав замену $y = \frac{dx}{dt}$:

$$\frac{dy}{dt} = -4y - 13x + e^{\sin(t)}, \quad \frac{dx}{dt} = y, \quad y(0.25) = 1, \quad x(0.25) = -1.$$

Составим функцию вычисления системы и решим ее так, как показано в листинге 8.5. График решения приведен на рис. 8.5.

```
function F=FF(t,x)
F=[-4*x(1)-13*x(2)+exp(t);x(1)];
endfunction
-->//Решение системы дифференциальных уравнений
-->X0=[1;-1];t0=0.25;t=0.25:0.05:2;
-->y=ode("stiff",X0,t0,t,FF);
-->//Вывод графика решения
-->plot(t,y)
```

Листинг 8.5

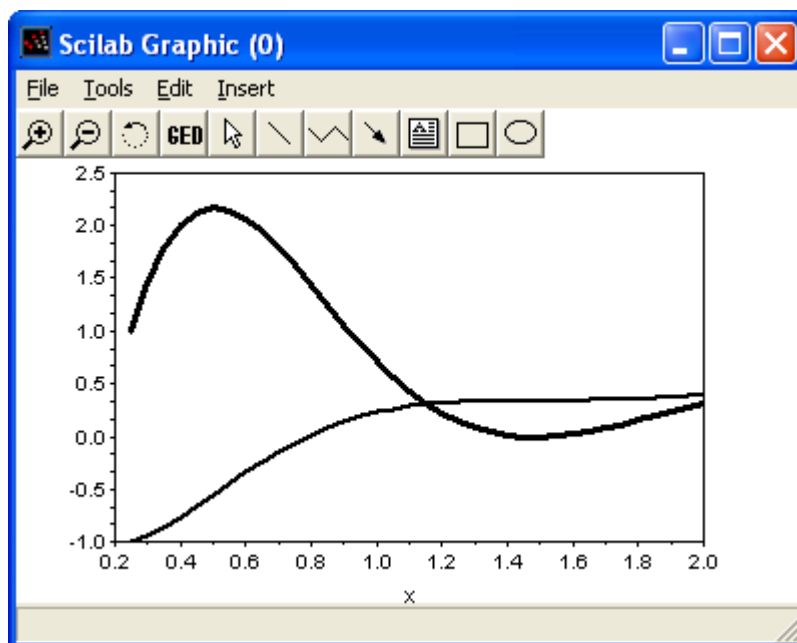
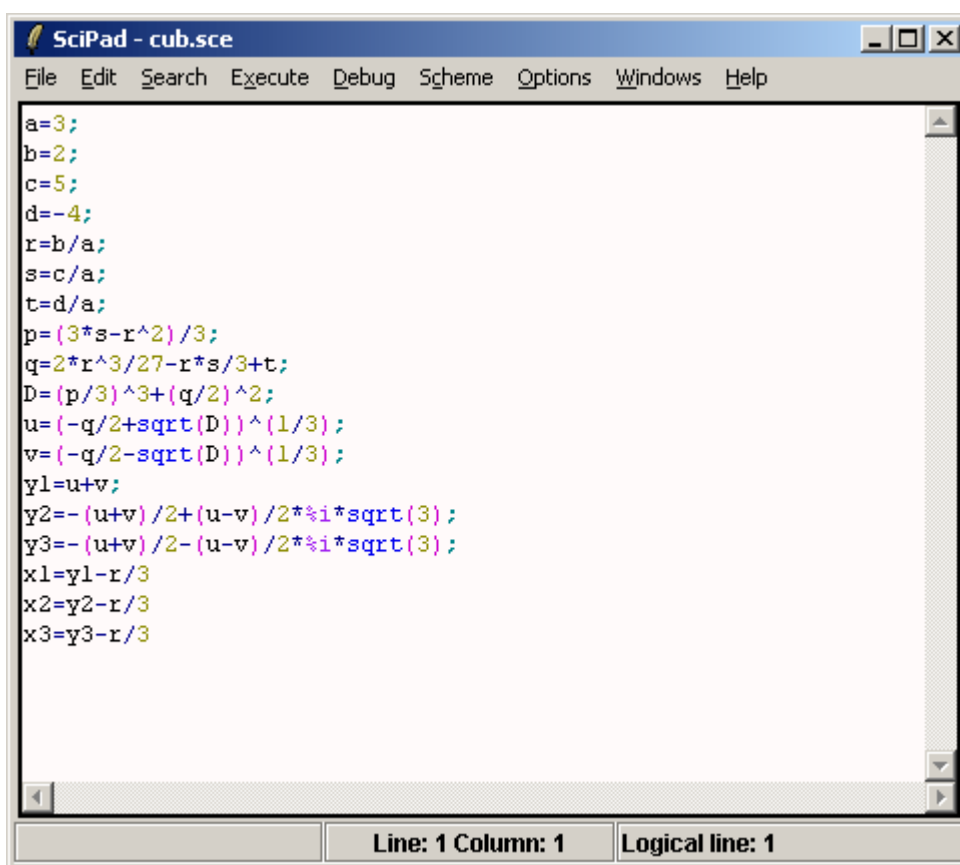


Рис.8.5. Решение задачи 8.5

9. Программирование в Scilab

Как уже рассматривалось ранее, работа в Scilab может осуществляться в режиме командной строки, но и в так называемом программном режиме. Напомним, что для создания программы (программу в Scilab иногда называют сценарием) необходимо:

1. Вызвать команду **Editor** из меню (см. рис. 9.1).
2. В окне редактора **SciPad** набрать текст программы.



```
SciPad - cub.sce
File Edit Search Execute Debug Scheme Options Windows Help
a=3;
b=2;
c=5;
d=-4;
r=b/a;
s=c/a;
t=d/a;
p=(3*s-r^2)/3;
q=2*r^3/27-r*s/3+t;
D=(p/3)^3+(q/2)^2;
u=(-q/2+sqrt(D))^(1/3);
v=(-q/2-sqrt(D))^(1/3);
y1=u+v;
y2=-(u+v)/2+(u-v)/2*i*sqrt(3);
y3=-(u+v)/2-(u-v)/2*i*sqrt(3);
x1=y1-r/3;
x2=y2-r/3;
x3=y3-r/3
Line: 1 Column: 1 Logical line: 1
```

Рис. 9.1. Окно *SciPad* с текстом программы

3. Сохранить текст программы с помощью команды **File-Save** в виде файла с расширением **sce**, например **file.sce**.
4. После чего программу можно будет вызвать набрав в командной строке **exec**, например **exec("file.sce")** или вызвав команду меню **File-Exec...**

Программный режим достаточно удобен, так как он позволяет сохранить разработанный вычислительный алгоритм в виде файла и повторять его при других исходных данных в других сессиях. Кроме обращений к функциям и операторов присваивания, в программных файлах могут использоваться операторы языка программирования Scilab (язык программирования Scilab будем называть sci-языком).

9.1. Основные операторы sci-языка

9.1.1. Функции ввода-вывода в Scilab

Для организации простейшего ввода в Scilab можно воспользоваться функциями

```
x=input('title');
```

или

```
x=x_dialog('title', 'stroka');
```

Функция **input** выводит в командной строке **Scilab** подсказку **title** и ждет пока пользователь введет значение, которое в качестве результата возвращается в переменную **x**. Функция **x_dialog** выводит на экран диалоговое окно (см. рис. 9.2), после чего пользователь может щелкнуть ОК и тогда **stroka** вернется в качестве результата в переменную **x**, либо ввести новое значение вместо **stroka**, которое и вернется в качестве результата в переменную **x**.

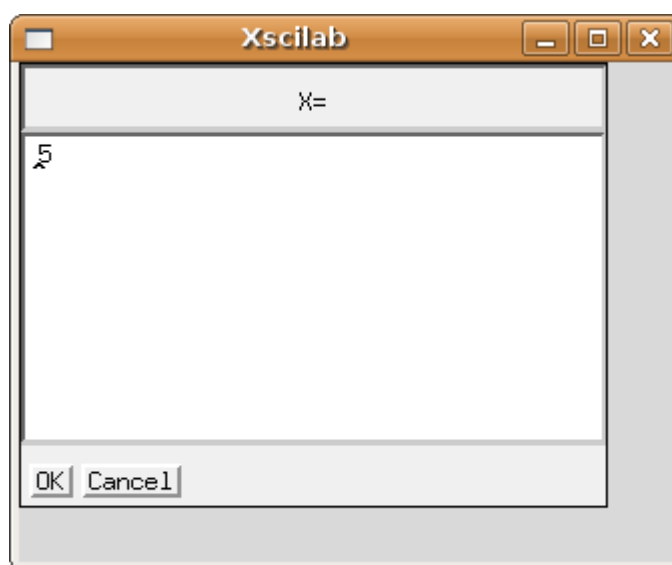


Рис. 9.2. Окно для ввода значения переменной

Функция **input** преобразовывает введенное значение к числовому типу данных, а функция **x_dialog** возвращает строковое значение. Поэтому при использовании функции **x_dialog** для ввода числовых значений, возвращаемую ею строку следует преобразовать в число с помощью функции **evstr**. Поэтому можно предложить следующую форму использования функции **x_dialog** для ввода числовых значений.

```
x=evstr(x_dialog('title', 'stroka'));
```

Для вывода в текстовом режиме можно использовать функцию **disp** следующей структуры **disp(b)**

Здесь **b** – имя переменной или заключенный в кавычки текст.

9.1.2. Оператор присваивания

Оператор присваивания имеет следующую структуру

a=b

здесь **a** – имя переменной или элемента массива, **b** – значение или выражение. В результате выполнения оператора присваивания переменной **a** присваивается значение выражения **b**.

9.1.3. Условный оператор

Одним из основных операторов, реализующим ветвление в большинстве языков программирования, является условный оператор **if**. Существует обычная и расширенная формы оператора **if** в **Scilab**. Обычный **if** имеет вид

```
if условие  
операторы1  
else  
операторы2  
end
```

Здесь **условие** – логическое выражение, **операторы1**, **операторы2** – операторы языка **Scilab** или встроенные функции. Оператор **if** работает по следующему алгоритму: если условие истинно, то выполняются **операторы1**, если ложно – **операторы2** (см. рис. 9.3).

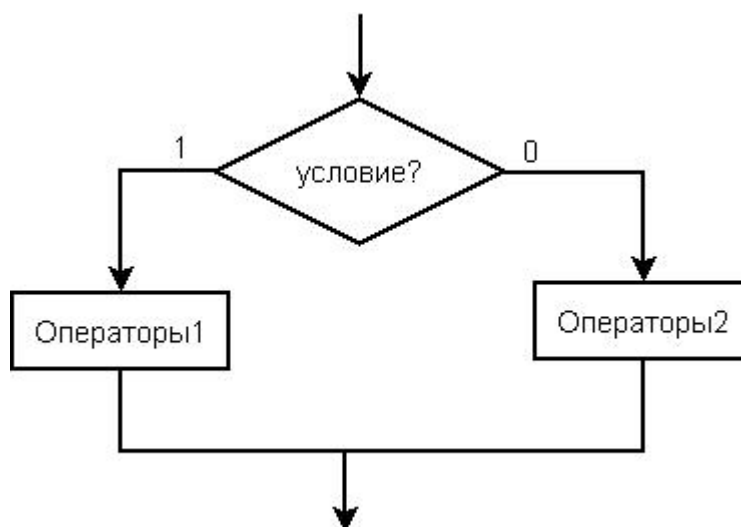


Рис. 9.3. Блок-схема условного оператора

В **Scilab** для построения логических выражений могут использоваться условные операторы: (**&**, **and** – логическое и, **|**, **or** – логическое или, **~**, **not** - логическое отрицание) и операторы отношения: **<** (меньше), **>** (больше), **==** (равно), **~=**, **<>** (не равно), **<=** (меньше или равно), **>=** (больше или равно).

Зачастую при решении практических задач недостаточно выбора выполнения или

невыполнения одного условия. В этом случае можно, конечно, по ветке **else** написать новый оператор **if**, но лучше воспользоваться расширенной формой оператора **if** (см. рис. 9.4).

```

if условие1
операторы1
elseif условие2
операторы2
elseif условие3
операторы3
...
elseif условие n
операторыn
else
операторы
end

```

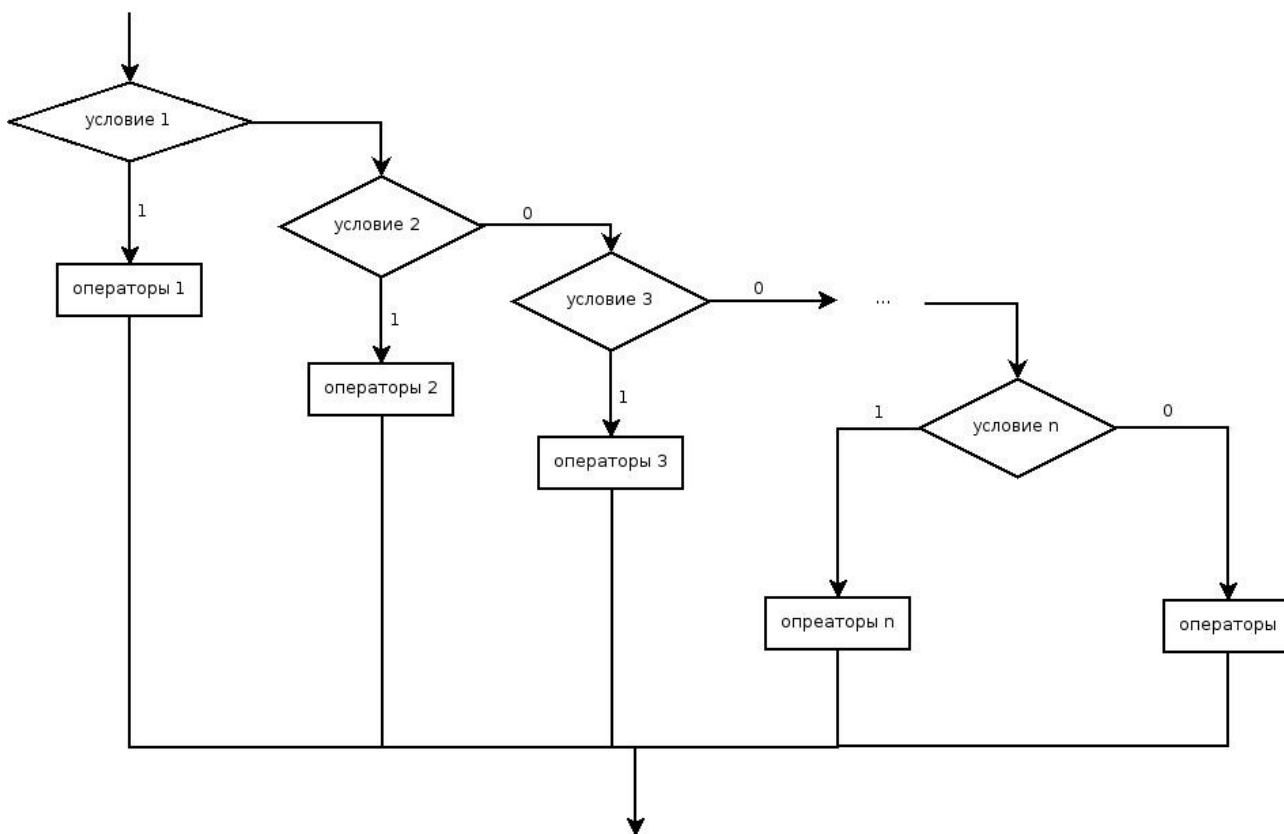


Рис. 9.4. Блок-схема оператора **if** в расширенной форме

В этом случае оператор **if** работает так: если **условие1** истинно, то выполняются **операторы1**, иначе проверяется **условие2**, если оно истинно, то выполняются **операторы2**, иначе проверяется **условие3** и т.д. Если ни одно из условий по веткам **else** и **elseif** не выполняется, то выполняются операторы по ветке **else**. На рис. 4. представлена блок-схема

расширенной формы оператора **if**.

ЗАДАЧА 9.1. В качестве примера программирования разветвляющегося процесса рассмотрим решение биквадратного уравнения $ax^4+bx^2+c=0$.

Для решения биквадратного уравнения необходимо заменой $y=x^2$ привести его к квадратному и решить это уравнение. Входными данными этой задачи являются коэффициенты биквадратного уравнения a, b, c . Выходными данными являются корни уравнения x_1, x_2, x_3, x_4 .

Алгоритм состоит из следующих этапов:

1. Ввод коэффициентов уравнения a, b и c .
2. Вычисление дискриминанта уравнения d .
3. Если $d < 0$, определяются y_1 и y_2 , а иначе корней нет.
4. Если $y_1, y_2 < 0$, то корней нет.
5. Если $y_1, y_2 < 0$, то вычисляются четыре корня по формулам и выводятся значения корней.
6. Если условия 4) и 5) не выполняются, то необходимо проверить знак y_1 .
7. Если y_1 неотрицательно, то вычисляются два корня по формуле $\pm\sqrt{y_1}$, иначе оба корня вычисляются по формуле $\pm\sqrt{y_2}$. Вычисленные значения корней выводятся.

Блок-схема решения этой задачи представлена на рис. 9.5, программа решения биквадратного уравнения на `sci`-языке приведена на листинге 9.1, ее вызов и результаты работы на листинге 9.2.

```
a=input('a=');
b=input('b=');
c=input('c=');
// Вычисляем дискриминант.
d=b*b-4*a*c;
// Если дискриминант отрицателен,
if d<0
// то вывод сообщения,
disp('Real roots are not present');
else
//иначе-вычисление корней соответствующего
// квадратного уравнения.
x1=(-b+sqrt(d))/2/a;
x2=(-b-sqrt(d))/2/a;
```

```
// Если оба корня отрицательны,  
if (x1<0)&(x2<0)  
// вывод сообщения об отсутствии действительных корней.  
disp('Real roots are not present');  
// иначе, если оба корня положительны,  
elseif (x1>=0)&(x2>=0)  
// вычисление четырех корней.  
disp('Four real roots');  
y1=sqrt(x1);  
y2=-y1;  
y3=sqrt(x2);  
y4=-y2;  
disp(y1,y2,y3,y4);  
//Иначе,если оба условия (x1<0)&(x2<0) и (x1>=0)&(x2>=0)  
// не выполняются,  
else  
// то вывод сообщения  
disp('Two real roots');  
// Проверка знака x1.  
if x1>=0  
//Если x1 положителен, то вычисление двух корней биквадратного  
// уравнения, извлечением корня из x1,  
y1=sqrt(x1);  
y2=-y1;  
disp(y1);  
disp(y2);  
// иначе (остался один вариант - x2 положителен),  
// вычисление двух  
// корней биквадратного уравнения извлечением корня из x2.  
else  
y1=sqrt(x2); y2=-y1;  
disp(y1); disp(y2);  
end  
end end
```

Листинг 9.1.

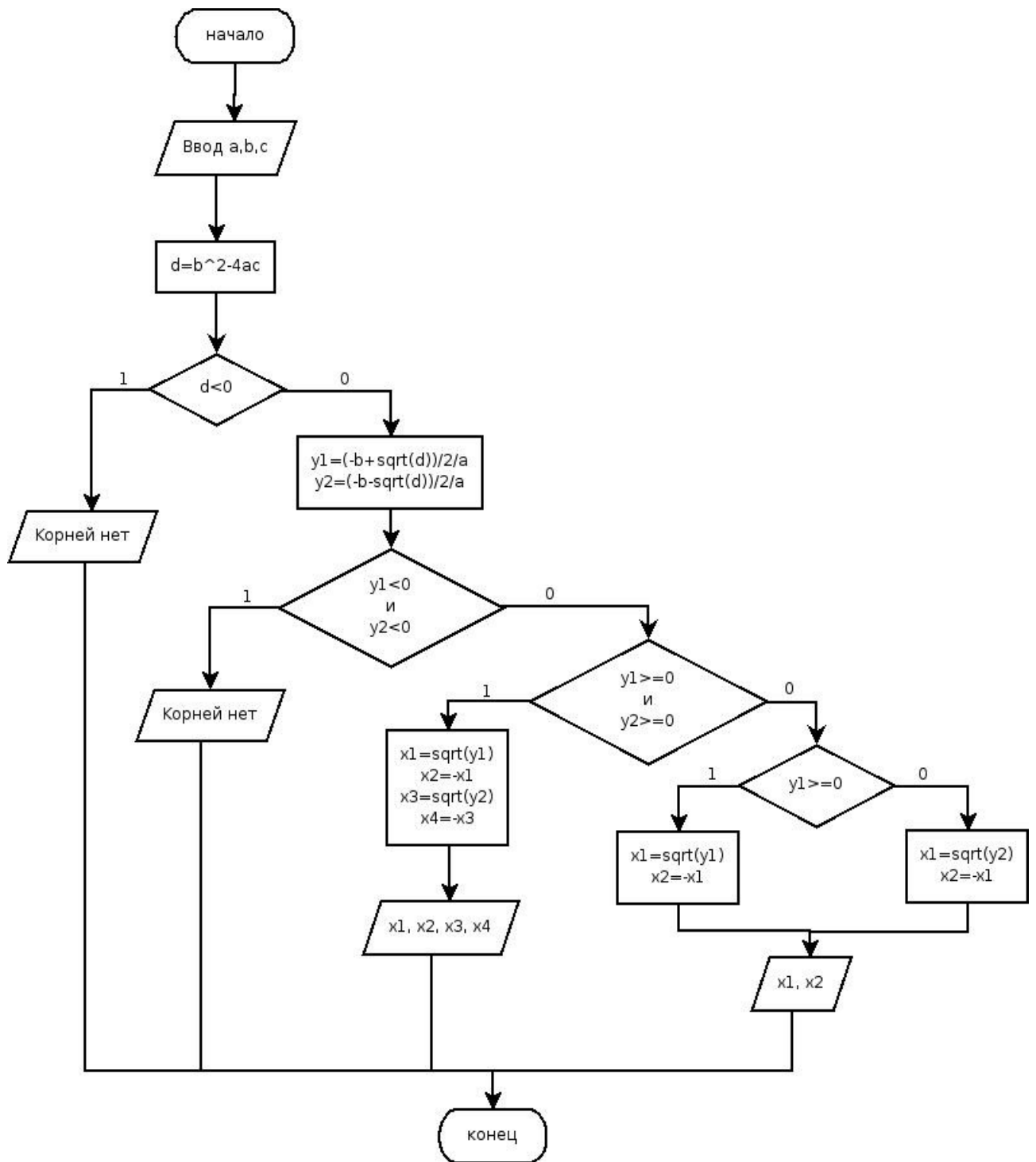


Рис. 9.5.

```

-->exec("G:/Lecture Scilab EG/2/l1.sci");
a--->-6
b--->9
c--->-1

Four real roots
0.3476307
    
```

```
1.1743734
- 0.3476307
0.3476307
```

Листинг 9.2.

9.1.4. Оператор альтернативного выбора

Еще одним способом организации разветвлений является оператор альтернативного выбора следующей структуры:

```
select параметр
case значение1 then операторы1
case значение2 then операторы2
...
else операторы
end
```

Оператор **select** работает следующим образом: если значение параметра равно значению1, то выполняются операторы1, иначе если параметр равен значению2, то выполняются операторы2; в противном случае, если значение параметра совпадает со значением3, то выполняются операторы3 и т.д. Если значение параметра не совпадает ни с одним из значений в группах **case**, то выполняются операторы, которые идут после служебного слова **else**.

Конечно, любой алгоритм можно запрограммировать без использования **select**, используя только **if**, но использование оператора альтернативного выбора **select** делает программу более компактной.

Рассмотрим использование оператора **select** на примере решения следующей задачи.

ЗАДАЧА 9.2. Вывести на печать название дня недели, соответствующее заданному числу **D**, при условии, что в месяце 31 день и 1-е число – понедельник.

Для решения задачи вычисляем остаток от деления числа x на k

$$x - \text{int}(x/k) * k$$

и условием, что 1-е число – понедельник. Если в результате остаток от деления заданного числа D на семь будет равен единице, то это понедельник, двойке – вторник, тройке – среда и так далее. Следовательно, при построении алгоритма необходимо использовать семь условных операторов.

Решение задачи станет значительно проще, если при написании программы воспользоваться оператором варианта (см. листинг 9.3). Вызов программы представлен на листинге 9.4.

```
D=input('Enter a number from 1 to 31');
//Вычисление остатка от деления D на 7, сравнение его с числами
от 0 до 6.
select D-int(D/7)*7
case 1 then disp('Monday');
case 2 then disp('Tuesday');
case 3 then disp('Wednesday');
case 4 then disp('Thursday');
case 5 then disp('Friday');
case 6 then disp('Saturday');
else
disp('Sunday');
end
```

Листинг 9.3.

```
-->exec('G:\Lecture Scilab EG\2\l2.sci');disp('exec done');
Enter a number from 1 to 31-->19
Friday
```

Листинг 9.4.

Рассмотрим операторы цикла в Scilab. В Sci-языке Scilab есть два вида цикла – оператор цикла с предусловием **while** и оператор **for**.

9.1.5. Оператор **while**

Оператор цикла **while** имеет вид

while условие

операторы

end

Здесь условие – логическое выражение; операторы будут выполняться циклически, пока логическое условие истинно. Блок-схема оператора **while** представлена на рис. 9.6.

Оператор цикла **while** обладает значительной гибкостью, но не слишком удобен для организации «строгих» циклов, которые должны быть выполнены заданное число раз. Оператор цикла **for** используется именно в этих случаях.

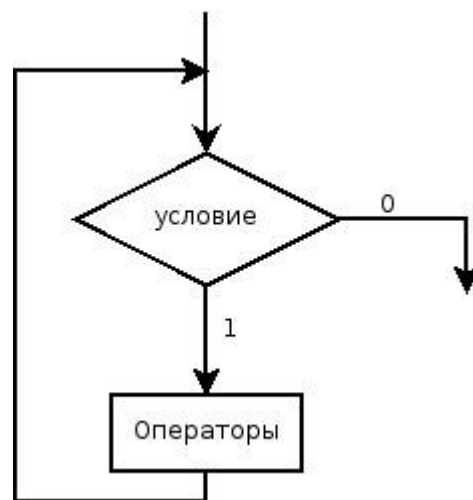


Рис. 9.6. Блок-схема оператора *while*

9.1.6. Оператор **for**

Оператор цикла **for** имеет вид

```
for x=xn:hx:xk
операторы
end
```

Здесь **x** – имя скалярной переменной – параметра цикла, **xn** – начальное значение параметра цикла, **xk** – конечное значение параметра цикла, **hx** – шаг цикла. Если шаг цикла равен 1, то **hx** можно опустить, и в этом случае оператор **for** будет таким.

```
for x=xn:xk
операторы
end
```

Выполнение цикла начинается с присвоения параметру стартового значения ($x=xn$). Затем следует проверка, не превосходит ли параметр конечное значение ($x>xk$). Если результат проверки утвердительный, то цикл считается завершенным, и управление передается следующему за телом цикла оператору. В противном случае выполняются операторы в цикле (тело цикла). Далее параметр меняет свое значение ($x=x+hx$). Далее снова производится проверка значения параметра цикла, и алгоритм повторяется (см. рис. 9.7).

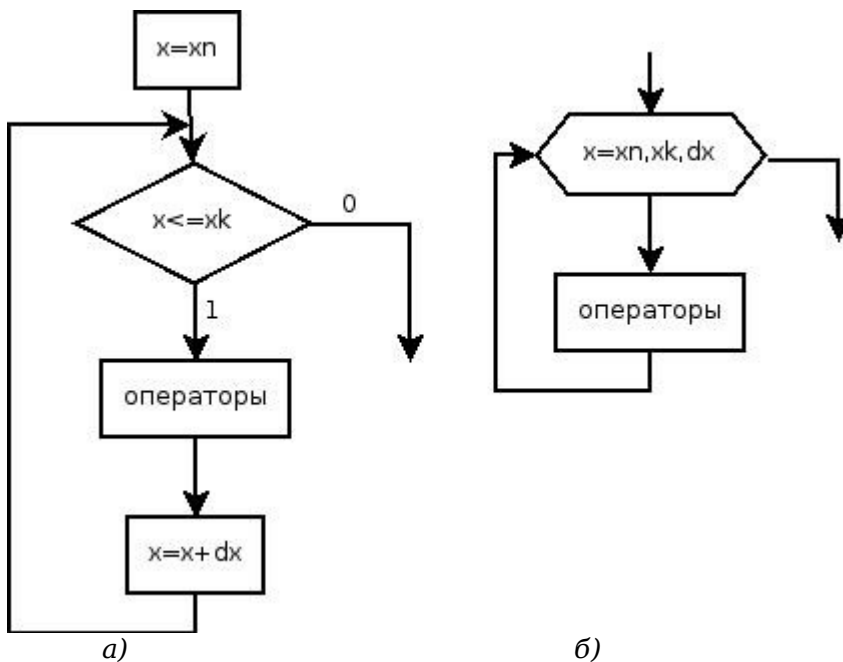


Рис. 9.7. Оператор for:

- а) блок-схема алгоритма оператора,
- б) стандартная блок-схема оператора

Особенностью программирования задач обработки массивов (одномерных, двумерных) на sci-языке является возможность как поэлементной обработки массивов (как в любом языке программирования), так и использование функций Scilab для работы массивами и матрицами.

Рассмотрим основные алгоритмы обработки массивов и матриц и их реализацию на sci-языке.

9.2. Обработка массивов и матриц в Scilab

9.2.1. Ввод-вывод массивов и матриц

Ввод массивов и матриц следует организовывать поэлементно, на рис. 9.8 приведена блок-схема алгоритма ввода элементов массивов, а на рис. 9.9 – матриц. На листингах 9.5, 9.6 приведены программы ввода элементов массивов и матриц на M-языке, реализующие эти алгоритмы.

```
N=input('N=');
disp('Vvod massiva x');
for i=1:N
x(i)=input('X=');
end
disp(x);
```

Листинг 9.5. Ввод элементов массива

```

N=input('N=');
M=input('M=');
disp('Vvod matrici');
for i=1:N
for j=1:M
a(i,j)=input('');
end
end
disp(a);

```

Листинг 9.6. Ввод элементов матрицы

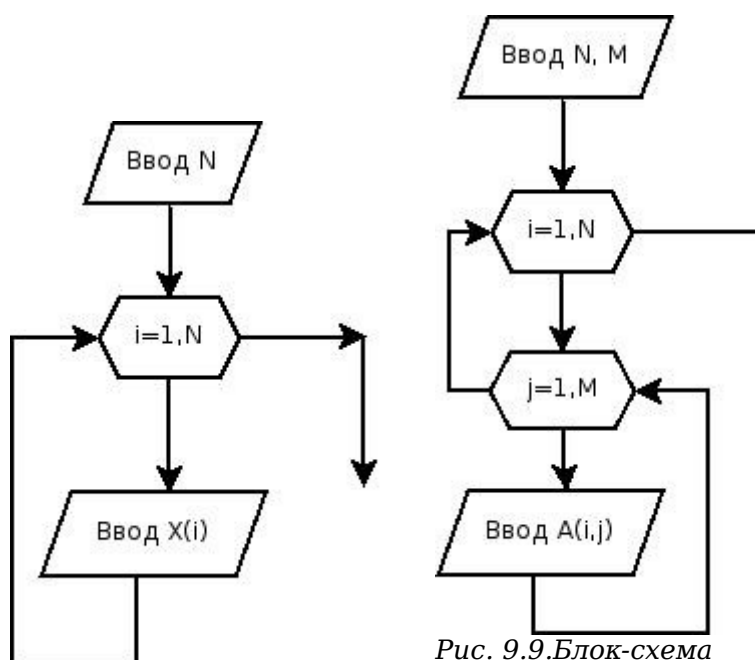


Рис. 9.8. Блок-схема ввода элементов массива

Рис. 9.9. Блок-схема ввода элементов матрицы

9.2.2. Вычисление суммы и произведения элементов массива (матрицы)

Алгоритм нахождения суммы состоит в следующем: вначале сумма равна 0 ($s=0$), затем к s добавляем первый элемент массива и результат записываем опять в переменную s , далее к переменной s добавляем второй элемент массива и результат записываем в s и далее аналогично добавляем к s остальные элементы массива. При нахождении суммы элементов матрицы последовательно суммируем элементы всех строк.

Алгоритм нахождения произведения следующий: на первом начальное значение произведения равно 1 ($p=1$), затем последовательно умножаем p на очередной элемент, и результат записываем в p .

На рис. 9.10, 9.11 приведены блок-схемы алгоритмов нахождения суммы и произведения элементов массива, на рис. 9.12, 9.13 – алгоритмы нахождения суммы и произведения элементов матрицы. На листингах 9.7-9.10 представлены элементы программ, реализующие эти алгоритмы.

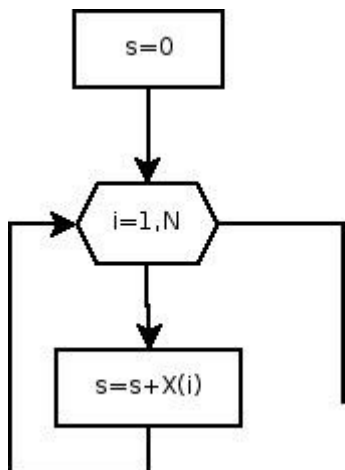


Рис. 9.10. Блок-схема алгоритма нахождения суммы элементов массива

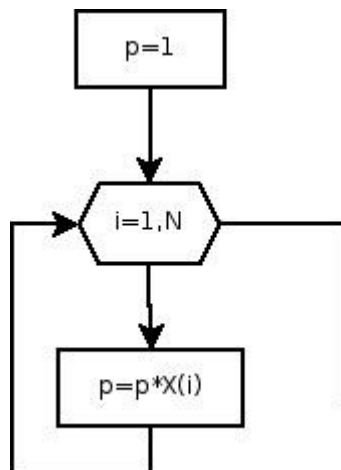


Рис. 9.11. Блок-схема алгоритма нахождения произведения элементов массива

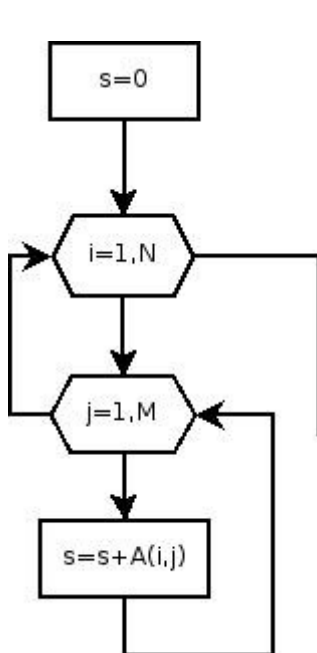


Рис. 9.12. Блок-схема нахождения суммы элементов матрицы

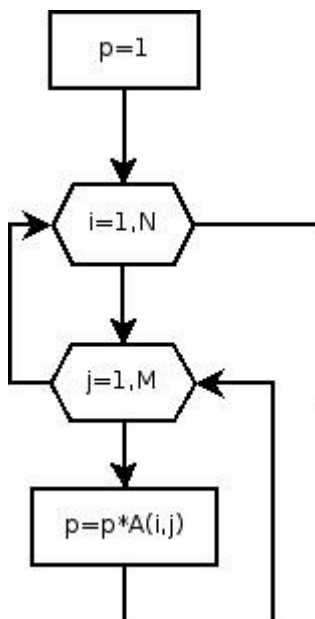


Рис. 9.13. Блок-схема нахождения произведения элементов матрицы

```
s=0;
for i=1:length(x)
s=s+x(i);
end
```

Листинг 9.7. Программа вычисления суммы элементов массива

```
p=1;
for i=1:length(x)
p=p*x(i);
end
```

Листинг 9.8. Программа вычисления произведения элементов массива

```
N=input('N=');
M=input('M=');
disp('Vvod matrici');
for i=1:N
for j=1:M
a(i,j)=input('');
end
end
disp(a);
s=0;
for i=1:N
for j=1:M
s=s+a(i,j);
end
end
disp(s);
```

Листинг 9.9. Программа вычисления суммы элементов матрицы

```
p=1;
for i=1:N
for j=1:M
p=p*a(i,j);
end
end
```

Листинг 9.10. Программа вычисления произведения элементов матрицы**9.2.3. Поиск максимального (минимального) элемента массива (матрицы)**

Пусть в переменной с именем Max хранится значение максимального элемента массива, а в переменной с именем Nmax – его номер. Алгоритм решения задачи поиска максимума и его номера в массиве следующий. Предположим, что первый элемент массива является

максимальным и запишем его в переменную Max , а в $Nmax$ – его номер (1). Затем все элементы, начиная со второго, сравниваем в цикле с максимальным. Если текущий элемент массива оказывается больше максимального, то записываем его в переменную Max , а в переменную $Nmax$ – текущее значение индекса i . Алгоритм нахождения максимального элемента в массиве приведен в блок-схеме на рис. 9.14.

На листинге 9.11 представлен фрагмент программы поиска максимума.

```
Max=a(1);
Nmax=1;
for i=2:N
if x(i)>Max
Max=x(i);
Nmax=i;
end;
end;
```

Листинг 9.11. Реализация алгоритма поиска максимума

Алгоритм поиска минимального элемента в массиве будет отличаться от приведенного выше лишь тем, что в условном блоке и, соответственно, в конструкции `if` текста программы знак поменяется с `>` на `<`. На рис. 9.15 представлена блок-схема поиска минимального элемента матрицы и его индексов: $Nmin$ – номер строки, $Lmin$ – номер столбца минимального элемента.

Обратите внимание, что при поиске минимального (максимального) элемента матрицы циклы по i и j начинаются с 1. Иначе при обработке элементов будет пропущена первая строка или первый столбец при сравнении a_{ij} с min . На листинге 9.12 представлена реализация этого алгоритма.

```
Min=a(1,1); Nmin=1; Lmin=1;
for i=1:N
for j=1:M
if a(i,j)<Min
Min=a(i,j);
Nmin=i;
Lmin=j;
end; end;
end;
```

Листинг 9.12. Программа поиска минимального элемента матрицы и его индексов

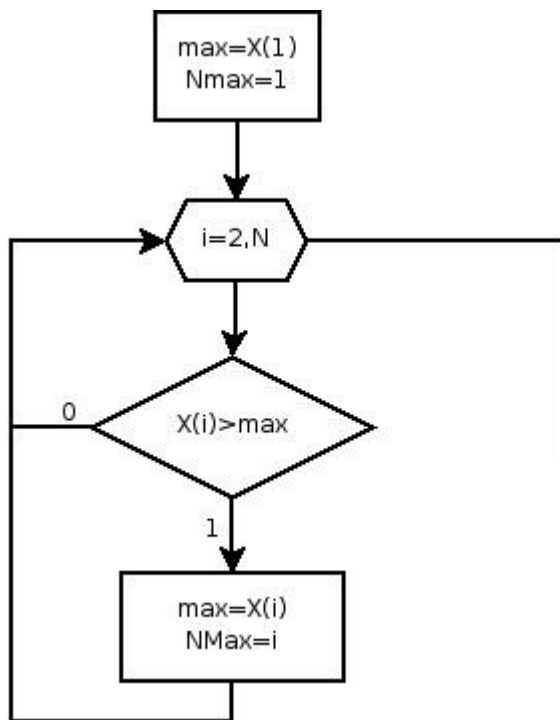


Рис. 9.14. Поиск минимального элемента в массиве и его номера

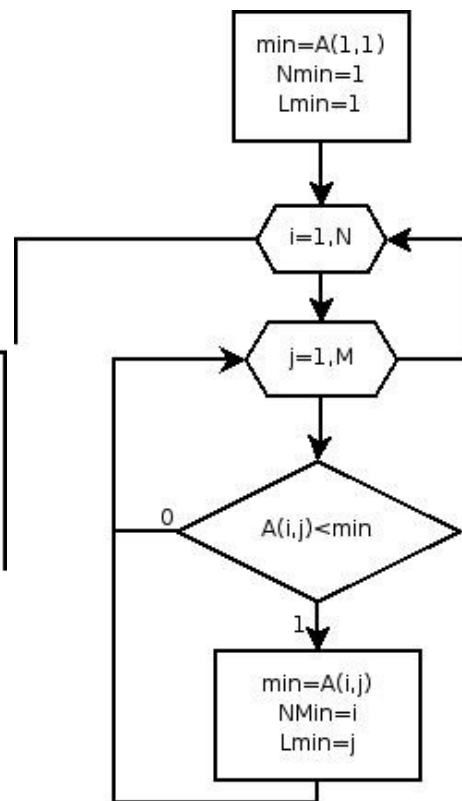


Рис. 9.15. Поиск минимального элемента матрицы и его индексов

9.2.4. Алгоритм упорядочивания элементов массива

Сортировка (упорядочивание) представляет собой процесс упорядочения элементов в массиве в порядке возрастания или убывания их значений. Например, массив X из n элементов будет отсортирован в порядке возрастания значений его элементов, если

$$X[1] \leq X[2] \leq \dots \leq X[n],$$

и в порядке убывания, если

$$X[1] \geq X[2] \geq \dots \geq X[n].$$

Рассмотрим наиболее известный алгоритм сортировки методом пузырька. Сравним первый элемент массива со вторым, если первый окажется больше второго, то поменяем их местами. Те же действия выполним для второго и третьего, третьего и четвертого, i -го и $(i+1)$ -го, $(n-1)$ -го и n -го элементов. В результате этих действий самый большой элемент станет на последнее (n) -е место. Теперь повторим данный алгоритм сначала, но последний (n) -й элемент рассматривать не будем, так как он уже занял свое место. После проведения данной операции самый большой элемент оставшегося массива станет на $(n-1)$ -е место. Так повторяем до тех пор, пока не упорядочим по возрастанию весь массив. Блок-схема алгоритма представлена на рис. 9.16.

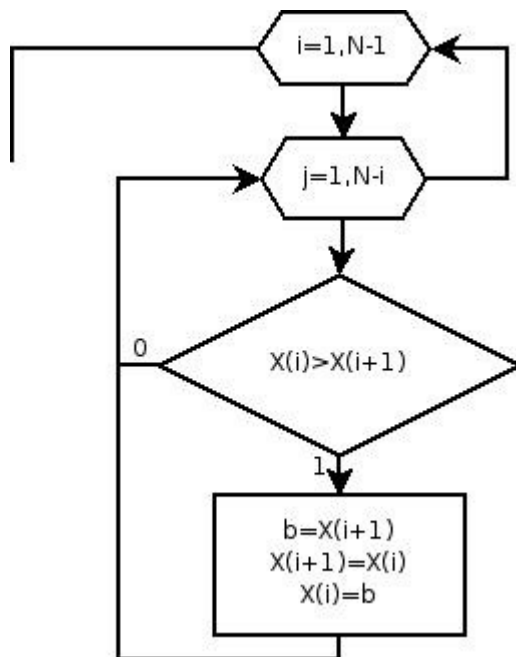


Рис. 9.16. Алгоритм упорядочивания массива по возрастанию

Алгоритм сортировки по убыванию будет отличаться от приведенного на рис. 9.16 заменой знака $>$ на $<$. На листинге 13 представлен фрагмент программы на scilab-языке, реализующий алгоритм сортировки по возрастанию.

```

x=[-3 5 7 49 -8 11 -5 32 -11];
for i=1:length(x)-1
for j=1:length(x)-i
if x(j)>x(j+1)
b=x(j);
x(j)=x(j+1);
x(j+1)=b;
end;
end;
end;
disp(x);
  
```

Листинг 9.13. Программа упорядочивания по возрастанию

9.2.5. Удаление элемента из массива

Необходимо удалить из массива x , состоящего из n элементов, m -й по номеру элемент. Для этого достаточно записать элемент $(m+1)$ на место элемента m , $(m+2)$ – на место $(m+1)$ и т.д., n – на место $(n-1)$ и при дальнейшей работе с этим массивом использовать $n-1$ элемент.

Алгоритм удаления из массива x размерностью n элемента с номером m приведен на рис. 9.17. На листинге 9.14 приведен фрагмент программы, реализующий этот алгоритм.

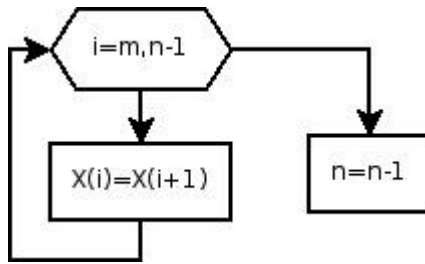


Рис. 9.17. Алгоритм удаления

```

x=[3 2 1 5 4 6 8 7];
disp(x);
n=length(x);
m=input('m=');
for i=m:n-1
x(i)=x(i+1);
end;
// Удаление n-го элемента из массива.
x(:,n)=[];
n=n-1;
disp(x);

```

Листинг 9.14. Программа удаления m -го элемента из массива $x(n)$

9.3. Работа с файлами в Scilab

Рассмотрим функции Scilab для работы с файлами.

9.3.1. Функция открытия файла `mopen`

Обращение к функции открытия файла имеет вид:

[fd,err]=mopen(file, mode)

file – строка, в которой хранится имя файла,

mode – режим работы с файлом:

- 'r' открываемый текстовый файл доступен для чтения,
- 'rb' открываемый двоичный файл доступен для чтения,
- 'w' – создаваемый пустой текстовый файл предназначен только для записи информации;
- 'wb' – создаваемый пустой двоичный файл предназначен только для записи

информации;

- 'a' – открываемый текстовый файл будет использоваться для добавления данных в конец файла; если файла нет, он будет создан;
- 'ab' – открываемый двоичный файл будет использоваться для добавления данных в конец файла; если файла нет, он будет создан;
- 'r+' – открываемый текстовый файл используется в режиме чтения и записи;
- 'rb+' – открываемый двоичный файл используется в режиме чтения и записи;
- 'w+' – создаваемый пустой текстовый файл предназначен для чтения и записи информации;
- 'wb+' – создаваемый пустой двоичный файл предназначен для чтения и записи информации;
- 'a+' – открываемый текстовый файл будет использоваться для добавления данных в конец файла и чтения данных; если файла нет, он будет создан;
- 'ab+' – открываемый двоичный файл будет использоваться для добавления данных в конец файла и чтения данных; если файла нет, он будет создан.

9.3.2. Функция записи в текстовый файла `fprintf`

Функция записи в текстовый файл `mfprintf` имеет вид

`mfprintf(f, s1, s2).`

Здесь `f` – идентификатор файла (значение идентификатора возвращается функцией `fopen`), `s1` – строка вывода, `s2` – список выводимых переменных.

В строке вывода вместо выводимых переменных указывается строка преобразования следующего вида:

`%[ширина][.точность]тип.`

Значения параметров строки преобразования приведены в таблице 9.1.

Таблица 9.1

Параметр	Назначение
Флаги	
-	Выравнивание числа влево. Правая сторона дополняется пробелами. По умолчанию выравнивание вправо.
+	Перед числом выводится знак «+» или «-»
Пробел	Перед положительным числом выводится пробел, перед отрицательным – «-»
#	Выводится код системы счисления: 0 –

Параметр	Назначение
	перед восьмеричным числом, 0x (0X) перед шестнадцатеричным числом.
Ширина	
n	Ширина поля вывода. Если n позиций недостаточно, то поле вывода расширяется до минимально необходимого. Незаполненные позиции заполняются пробелами.
0n	То же, что и n, но незаполненные позиции заполняются нулями.
Точность	
ничего	Точность по умолчанию
n	Для типов e, E, f выводить n знаков после десятичной точки
Модификатор	
h	Для d, i, o, u, x, X короткое целое
l	Для d, i, o, u, x, X длинное целое
Тип	
c	При вводе символьный тип char, при выводе один байт.
d,i	Десятичное со знаком
i	Десятичное со знаком
o	Восьмеричное int unsigned
u	Десятичное без знака
x, X	Шестнадцатеричное int unsigned, при x используются символы a-f, при X – A-F.
f	Значение со знаком вида [-]dddd.dddd
e	Значение со знаком вида [-]d.dddde[+ -]ddd
E	Значение со знаком вида [-]d.ddddE[+ -]ddd
g	Значение со знаком типа e или f в зависимости от значения и точности
G	Значение со знаком типа E или F в зависимости от значения и точности
s	Строка символов

В строке вывода могут использоваться некоторые специальные символы, приведенные в табл. 9.2.

Таблица 9.2.

Некоторые специальные символы

Символ	Назначение
\b	Сдвиг текущей позиции влево
\n	Перевод строки
\r	Перевод в начало строки, не переходя на новую строку
\t	Горизонтальная табуляция
\'	Символ одинарной кавычки
\''	Символ двойной кавычки
\?	Символ ?

9.3.3. Функция чтения данных из текстового файла `mfscanf`

При считывании данных из файла можно воспользоваться функцией `mfscanf` следующего вида

$$A = \text{mfscanf}(f, s1)$$

Здесь `f` – идентификатор файла, который возвращается функцией `mopen`, `s1` – строка форматов вида

%[ширина][.точность]тип

Функция **`mfscanf`** работает следующим образом: из файла с идентификатором **`f`** считываются в переменную **`A`** значения в соответствии с форматом **`s1`**. При чтении числовых значений из текстового файла следует помнить, что два числа считаются разделенными, если между ними есть хотя бы один пробел, символ табуляции или символ перехода на новую строку.

При считывании данных из текстового файла пользователь может следить, достигнут ли конец файла с помощью функции **`meof(f)`** (**`f`** – идентификатор файла), которая возвращает единицу, если достигнут конец файла, и ноль в противном случае.

9.3.4 Функция закрытия файла `fclose`

После выполнения всех операций с файлом он должен быть закрыт с помощью функции `fclose` следующей структуры

`fclose(f)`

Здесь `f` – идентификатор закрываемого файла. С помощью функции `fclose('all')` можно

закрывать сразу все открытые файлы, кроме стандартных системных файлов.

Пример создания текстового файла приведен на листинге 9.15.

```
N=3;
M=4;
A=[2 4 6 7; 6 3 2 1; 11 12 34 10];
f=fopen('E:\abc.txt','w');
fprintf(f,'%d\t%d\n',N,M);
for i=1:N
for j=1:M
fprintf(f,'%g\t',A(i,j));
end
fprintf(f,'\n');
end
fclose(f);
```

Листинг 9.15. Создание текстового файла

Созданный текстовый файл можно увидеть на рис. 9.18

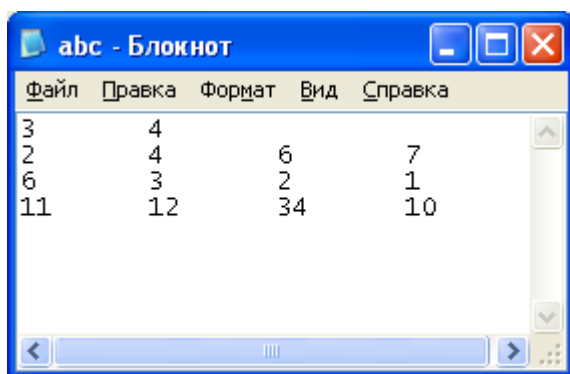


Рис. 9.18. Тестовый файл

Пример чтения данных из этого текстового файла приведен на листинге 9.16.

```
f=fopen('E:\abc.txt','r');
N=mfscanf(f,'%d');
M=mfscanf(f,'%d');
for i=1:N
for j=1:M
A(i,j)=mfscanf(f,'%g');
end
end
fclose(f);
```

Листинг 9.16. Чтение из текстового файла

Результаты работы файла сценаия, представленного на листинг 9.16 представлены на листинге 9.17.

```
N =
3.
M =
4.
A =
! 2. 4. 6. 7. !
! 6. 3. 2. 1. !
! 11. 12. 34. 10. !
```

Листинг 9.17. Результат работы файла сценария

9.4. Пример программы в Scilab

В качестве примера рассмотрим следующую задачу.

ЗАДАЧА 9.3. Положительные числа из массива Y переписать в массив X, удалить из массива X элементы, меньшие среднего арифметического и расположенные после минимального элемента.

Блок -схема решения задачи представлена на рис. 9.19, программа – на листинге 9.18. Использование переменной min1 в программе обусловлено тем, что в Scilab есть встроенная функция min. Использование переменной с именем min не позволит использовать встроенную функцию min.

В Scilab несложно оформлять личные функции.

```
N=input('N=');
disp('Vvod massiva Y');
for i=1:N
y(i)=input('Y=');
end
disp(y); k=0;
for i=1:N
if y(i)>0
k=k+1;
x(k)=y(i);
end; end;
```

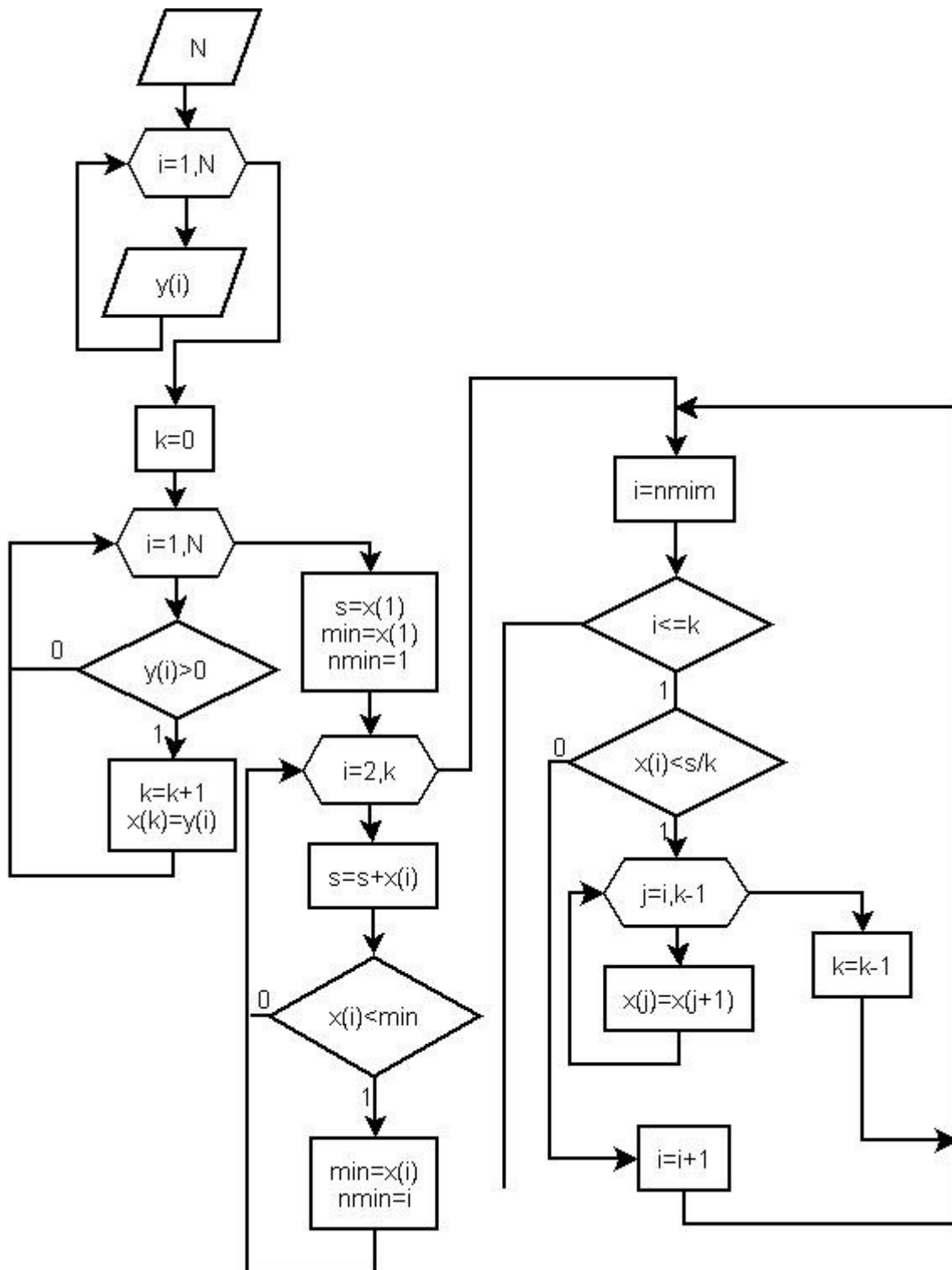


Рис. 9.19. Блок-схема решения задачи 9.3

```

disp(x);
s=x(1);
min1=x(1);
Nmin=1;
for i=2:k

```

```

s=s+x(i);
if x(i)<min1
min1=x(i);
Nmin=i;
end;
end;
i=Nmin;
while i<=k
if x(i)<s/k
for j=i:k-1
x(j)=x(j+1);
end;
//Удаление последнего элемента массива
x(k)=[];
k=k-1;
else
i=i+1;
end;
end;
disp(x);

```

Листинг 9.18. Решение задачи 9.1

9.5. Функции в Scilab

Структура функции в Scilab

```
function [y1,y2,...,yn]=ff(x1,x2,...,xm)
```

операторы

```
endfunction
```

Здесь **x1, x2, ..., xm** - список входных параметров функции; **y1,y2, ..., yn** - список выходных параметров функции.

Если вызываемая функция находится не в текущем файле, то перед ее вызовом следует загрузить файл, в котором находится функция **exec('file',-1)**.

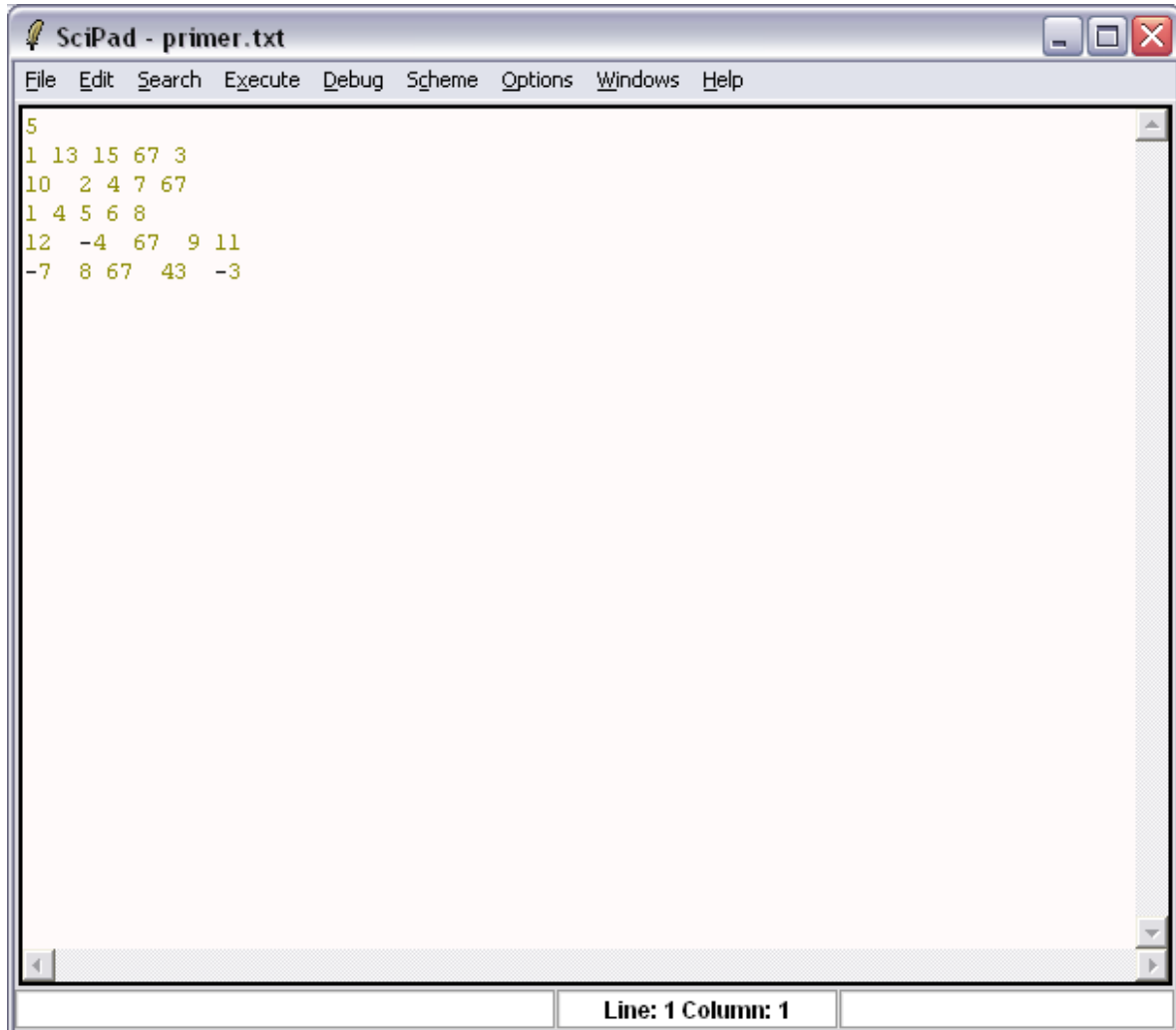
Рассмотрим пример с использованием функций и файлов.

ЗАДАЧА 9.4. В матрице $A(N,N)$ найти сумму элементов, расположенных на диагоналях матрицы, вычислить количество элементов, равных максимальному элементу матрицы. Число **N** и матрица **A** хранятся в текстовом файле **primer.txt** (см. рис. 9.20).

Для нахождения суммы, максимума и количества максимумов оформим функцию, заголовок ее будет иметь вид:

function [summa, maximum, kolichestvo]=matrica_A(N,N)

Блок-схема алгоритма приведена на рис.9.21, текст функции на листинге 9.19.



```
5
1 13 15 67 3
10 2 4 7 67
1 4 5 6 8
12 -4 67 9 11
-7 8 67 43 -3
```

Line: 1 Column: 1

Рис.9. 20. Содержимое текстового файла для задачи 9.2.

```
function [summa, maximum, kolichestvo]=matrica_A(A,N)
summa=0;
for i=1:N
summa=summa+A(i,i)+A(i,N+1-i);
end
if (N-int(N/2)*2)==1
summa=summa-A(int(N/2)*2+1,int(N/2)*2+1);
end
maximum=A(1,1);kolichestvo=1;
for i=1:N
```

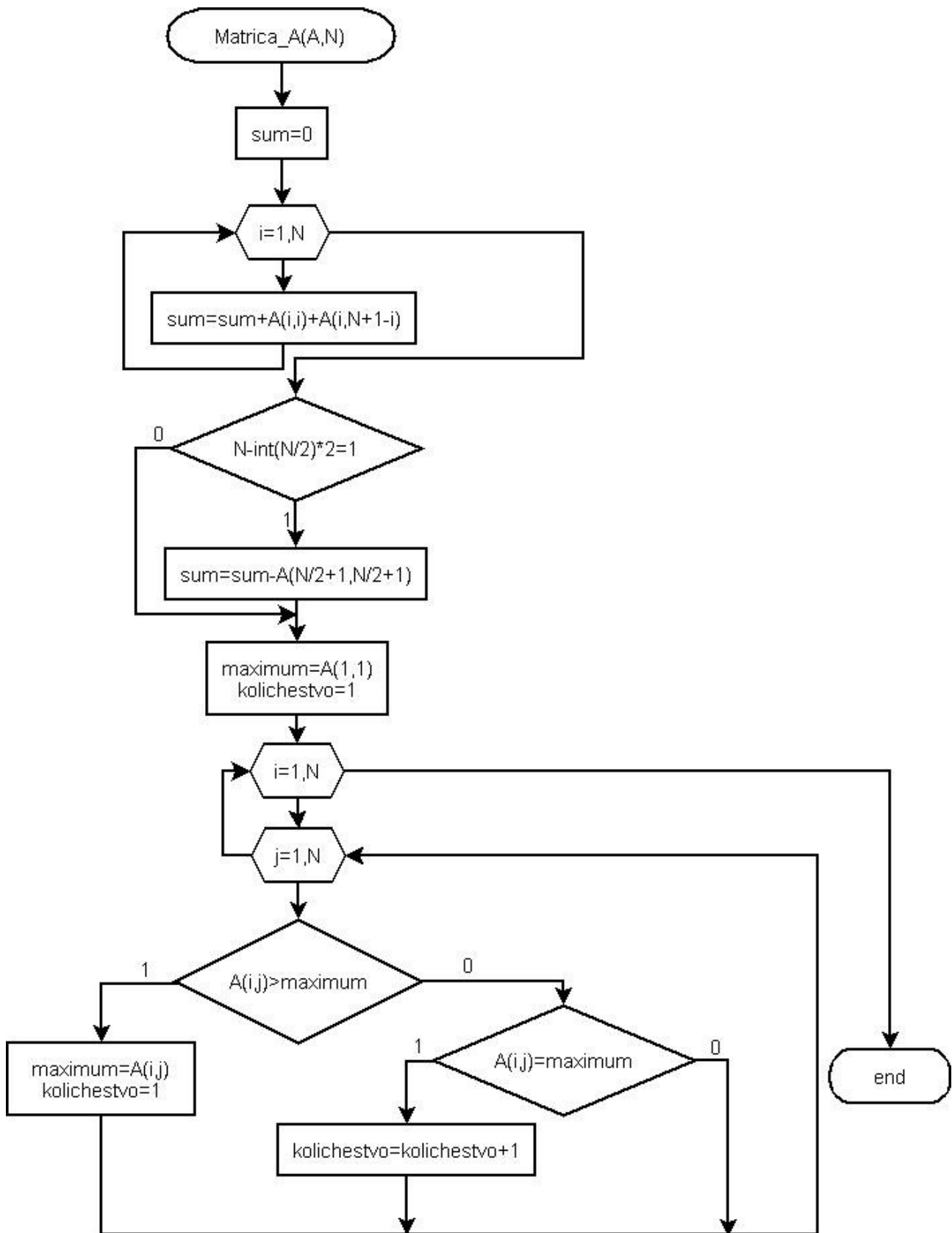


Рис. 9.21. Блок-схема функции *matrica_A*

for j=1:N

if A(i,j)>maximum

```
maximum=A(i,j);
kolichestvo=1;
elseif A(i,j)==maximum
kolichestvo=kolichestvo+1;
end
end
end
endfunction
```

Листинг 9.19. Текст функции `matrica_A`

Текст функции, предназначенной для чтения матрицы из файла и вызова функции `matrica_A` приведен на листинге 9.20, результаты на листинге 9.21.

```
f=fopen('G:\primer.txt','r');
N=mfscanf(f,'%d');
for i=1:N
for j=1:N
B(i,j)=mfscanf(f,'%g');
end
end
fclose(f);
[s,m,k]=matrica_A(B,N);
```

Листинг 9.20. Вызов функции `matrica_A`

```
-->exec("matrica_2.sci");
-->s
s =
    21.
-->m
m =
    67.
-->k
k =
    4.
```

Листинг 9.21. Результаты работы программы

В этой части были рассмотрены возможности программирования в Scilab. В следующей главе будут рассмотрены возможности построения визуальных программ.

10. Создание графических приложений в среде Scilab

Scilab позволяет создавать не только обычные программы для автоматизации расчетов, но и визуальные приложения, которые будут запускаться в среде Scilab. Основным объектом в среде Scilab является графическое окно.

10.1 Работа с графическим окном

Для создания пустого графического окна служит функция `figure`.

```
F=figure();
```

В результате выполнения этой команды будет создано данное графическое окно с именем `objfigure1`. По умолчанию первое окно получает имя `objfigure1`, второе – `objfigure2` и т.д. Указатель на графическое окно¹ записывается в переменную `F`. Размер и положение окна на экране монитора можно задавать с помощью параметра `'position'`, `[x y dx dy]`, где

- `x`, `y` - положение верхнего левого угла окна (по горизонтали и вертикали соответственно) относительно верхнего левого угла экрана;
- `dx` - размер окна по горизонтали (ширина окна) в пикселях;
- `dy` - размер окна по вертикали (высота окна) в пикселях.

Параметры окна можно задавать одним из двух способов.

1. Непосредственно при создании графического окна задаются его параметры. В этом случае обращение к функции `figure` имеет вид

```
F=figure('Свойство1', 'Значение1', 'Свойство2', 'Значение2', ..., 'Свойствоn', 'Значениеn')
```

здесь `'Свойство1'` – название первого параметра, `Значение1` – его значение, `'Свойство2'` – название второго параметра, `Значение2`² – значение второго параметра и т.д.

Например, с помощью команды

```
F=figure('position', [10 100 300 200]);
```

будет создано окно, представленное на рис. 10.1.

2. После создания графического окна с помощью функции `set(f, 'Свойство', 'Значение')` устанавливается значение параметров, здесь `f` – указатель на графическое окно, `'Свойство'` – имя параметра, `'Значение'` – его значение.

¹ Под указателем мы будем понимать переменную, в которой хранится адрес окна или другого объекта.

² `Значениеi` – будет в кавычках, если значением будет строка и без кавычек, если число.

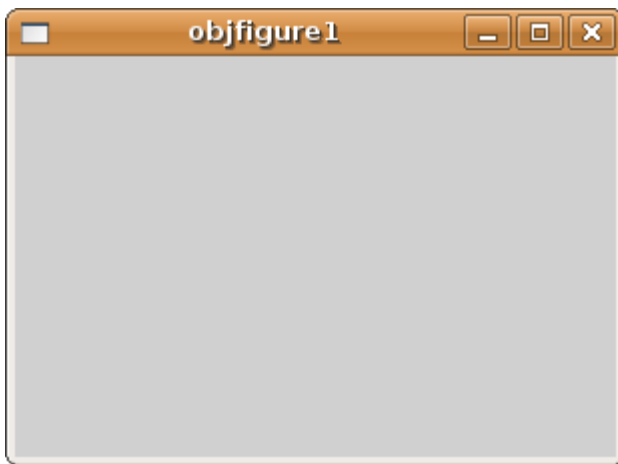


Рис. 10.1. Графическое окно

Например, следующие две строки (см. листинг 10.1) задают месторасположение и размер окна (см. рис. 10.2).

```
f=figure();  
set(f,'position',[20,40,600,450])
```

Листинг 10.1

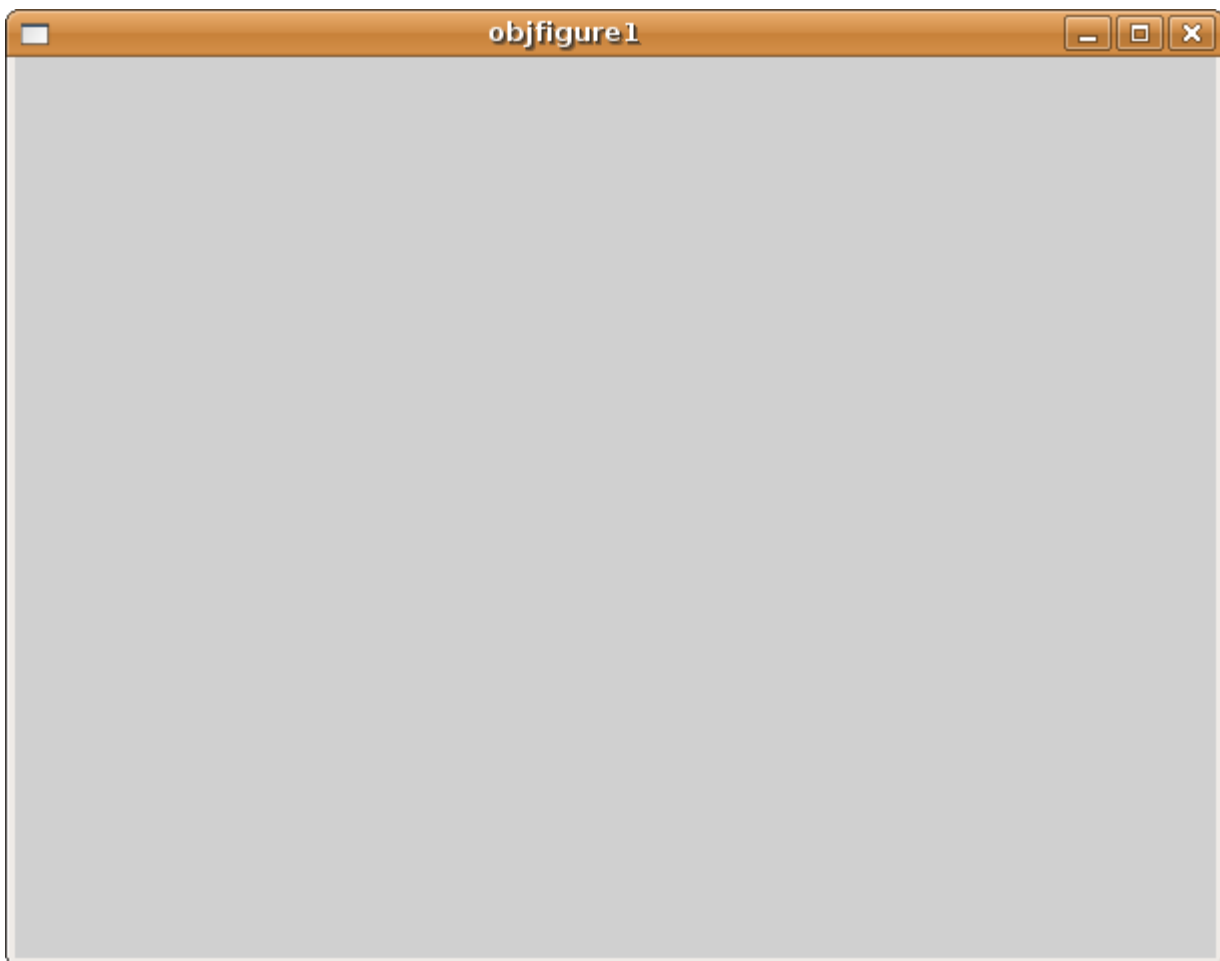


Рис. 10.2. Окно, созданное в результате выполнения листинга 10.1

Для изменения заголовка окна используется параметр `'figure_name'`, `'name'` определяющий заголовок окна (`'name'`). На листинге 10.2 приведен пример создания окна с именем **FIRST WINDOWS** (см. рис. 10.3). С помощью программы

```
f=figure();  
set(f,'position',[20,40,600,450]);  
set(f,'figure_name','FIRST WINDOW');
```

Листинг 10.2. Создание окна с именем FIRST WINDOW

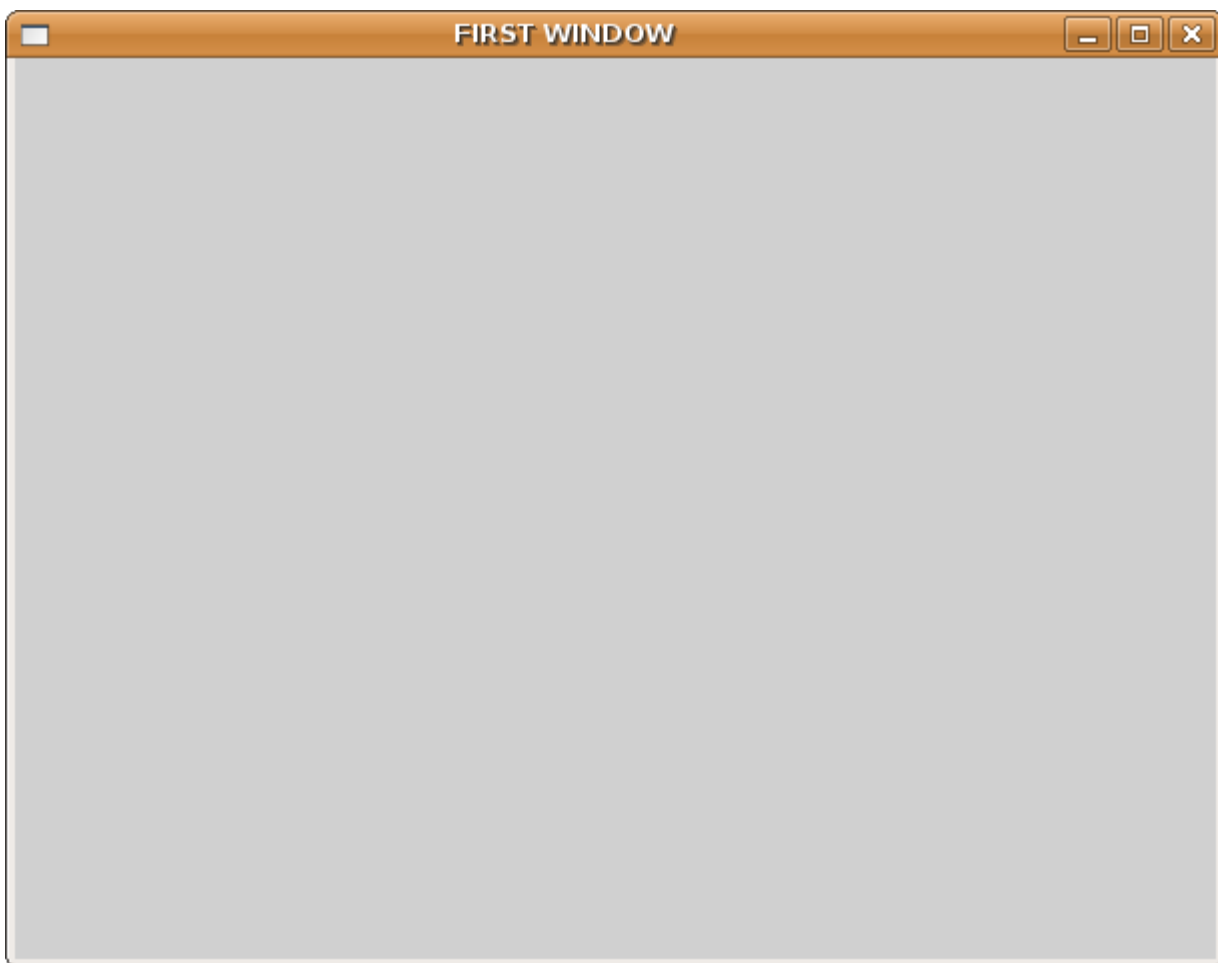


Рис.10. 3. Графическое окно с именем FIRST WINDOWS

Это же окно получить с помощью одной строки

```
f=figure('position',[20,40,600,450];'figure_name','FIRST WINDOW');
```

Графическое окно можно закрыть с помощью функция `close(f)` (здесь `f` – указатель на окно). Удаляется окно с помощью функции `delete(f)`, где `f`– указатель на окно.

10.2 Динамическое создание интерфейсных элементов. Описание основных функций

В Scilab используется динамический способ создания интерфейсных компонентов. Он заключается в том, что на стадии выполнения программы могут создаваться (и удаляться) те или

иные графические объекты (кнопки, метки, флажки и т.д.) и их свойствам присваиваются соответствующие значения.

Для создания любого интерфейсного компонента с заданными свойствами используется функция `uicontrol`, возвращающая указатель на формируемый компонент:

```
C=uicontrol(F, 'Style', 'тип_компонента', 'Свойство_1', Значение_1,
'Свойство_2', Значение_2, ... 'Свойство_k', Значение_k);
```

Здесь

- `C` – указатель на создаваемый компонент;
- `F` – указатель на объект, внутри которого будет создаваться компонент; первый аргумент функции `uicontrol` не является обязательным, и если он отсутствует, то родителем (владельцем) создаваемого компонента является текущий графический объект – текущее графическое окно;
- `'Style'`- служебная строка `Style`, указывает на стиль создаваемого компонента(символьное имя);
- `'тип_компонента'`- определяет, к какому классу принадлежит создаваемый компонент, это может быть `PushButton`, `Radiobutton`, `Edittext`, `StaticText`, `Slider`, `Panel`, `Button Group`, `Listbox` или др компоненты;
- `'Свойство_k'`, `Значение_k` – определяют свойства и значения отдельных компонентов, они будут описаны ниже конкретно для каждого компонента.

У существующего интерфейсного объекта можно изменить те или иные свойства с помощью функции `set`:

```
set(C, 'Свойство_1', Значение_1, 'Свойство_2', Значение_2, ...,
'Свойство_k', Значение_k)
```

- `C` – указатель на динамический компонент, параметры которого будут меняться; `C` может быть и вектором динамических элементов, в этом случае функция `set` будет задавать значения свойств для всех объектов `C(i)`;
- `'Свойство_k'`, `Значение_k`- изменяемые параметры и их значения.

Получить значение параметра компонентов можно с помощью функции `get` следующей структуры:

```
get(C, 'Свойство')
```

Здесь

- `C` – указатель на динамический интерфейсный компонент, значение параметра которого необходимо узнать;
- `'Свойство'`- имя параметра, значение которого нужно узнать.

Функция возвращает значение параметра. Далее мы поговорим об особенностях создания различных компонентов.

10.2.1 Командная кнопка

Командная кнопка типа `PushButton` создается с помощью функции `uicontrol`, в которой параметру `'style'` необходимо присвоить значение `'pushbutton'`. По умолчанию она не снабжается никакой надписью, имеет серый цвет и располагается в левом нижнем углу фигуры, Надпись на кнопке можно установить с помощью свойства `String` (см. листинг 10.3 и рис 10.4).

```
d=figure();
dbt=uicontrol(d,'Style','pushbutton');
set(dbt,'String','YES');
```

Листинг 10.3. Создание окна с кнопкой



Рис. 10.4. Окно с кнопкой

Модифицируем процедуру создания кнопки, задав дополнительно значения некоторых свойств: **Заголовок окна**, **надпись на кнопке** и ее **месторасположение**. Определяем позицию графического окна (левый верхний угол находится в точке с координатами (0,0)), ширина окна 250, а длина 100 пикселей. Аналогично определяются позиция и размеры кнопки с помощью параметра `position`, параметр `'string'` указывает имя динамического компонента (`'Button'`). Текст программы приведен на рис. 10.4, а на рис. 10.5 можно увидеть окно, которое получилось в результате работы этой программы.

```
f=figure();
set(f,'position',[0,0,250,100])
set(f,'figure_name','Button_example');
Button=uicontrol('style','pushbutton','string','Button',
'position',[50,50,100,20]);
```

Листинг 10.4. Определение свойств кнопки

Теперь при щелчке на кнопке вокруг ее надписи появляется пунктирный прямоугольник, свидетельствующий о том, что кнопка "находится в фокусе". Щелчок за пределами поверхности кнопки выведет ее из фокуса, и пунктирная рамка пропадет. Главным назначением командной кнопки является вызов функции, реагирующей на щелчок по кнопке.



Рис. 10.5 Кнопка с заданными свойствами

Щелчок генерирует событие `callback`, которое указывается как параметр функции `uicontrol`, после чего в одинарных кавычках указывается имя вызываемой по щелчку функции (обработчика события `CallBack`).

```
Button=uicontrol('style','pushbutton','string','Button',
'Callback','Function');
```

Здесь `Function` – имя вызываемой при наступлении события `callback` функции.

В качестве примера рассмотрим окно с кнопкой, при щелчке по которой появляется окно с графиком функции $y=\sin(x)$ (см. листинг 10.5). После запуска этой программы появится окно, представленное на рис. 10.6, при щелчке по кнопке `Button` вызывается обработчик события функция `gr_sin`, в результате чего появится окно, изображенное на рис. 10.7.

```
f=figure();
set(f,'position',[0,0,250,100])
set(f,'figure_name','Grafik');
Button=uicontrol('style','pushbutton','string','Button',...
'position',[50,50,100,20],'CallBack','gr_sin');
function y=gr_sin()
x=-5:0.2:5;
y=sin(x);
plot(x,y);
xgrid();
endfunction
```

Листинг 10.5. Пример кнопки с обработчиком события `CallBack`



Рис. 10.6. Окно с кнопкой

10.2.2. Метка

Следующим наиболее часто используемым компонентом является метка – текстовое поле для отображения текстовой информации. Для определения метки значения параметра 'Style' в функции `uicontrol` должно иметь значение 'text'. Компонент предназначен для вывода символьной строки (или нескольких строк). Выводимый текст - значение параметра 'string' может быть изменен только из программы. Рассмотрим пример создания текстового поля (метки) с помощью функции `uicontrol` (см. листинг 10.6 и рис. 10.7):

```
f=figure();  
uicontrol('Style','text','Position',[10,130,150,20],'String',  
'Метка');
```

Листинг 10.6. Создание метки

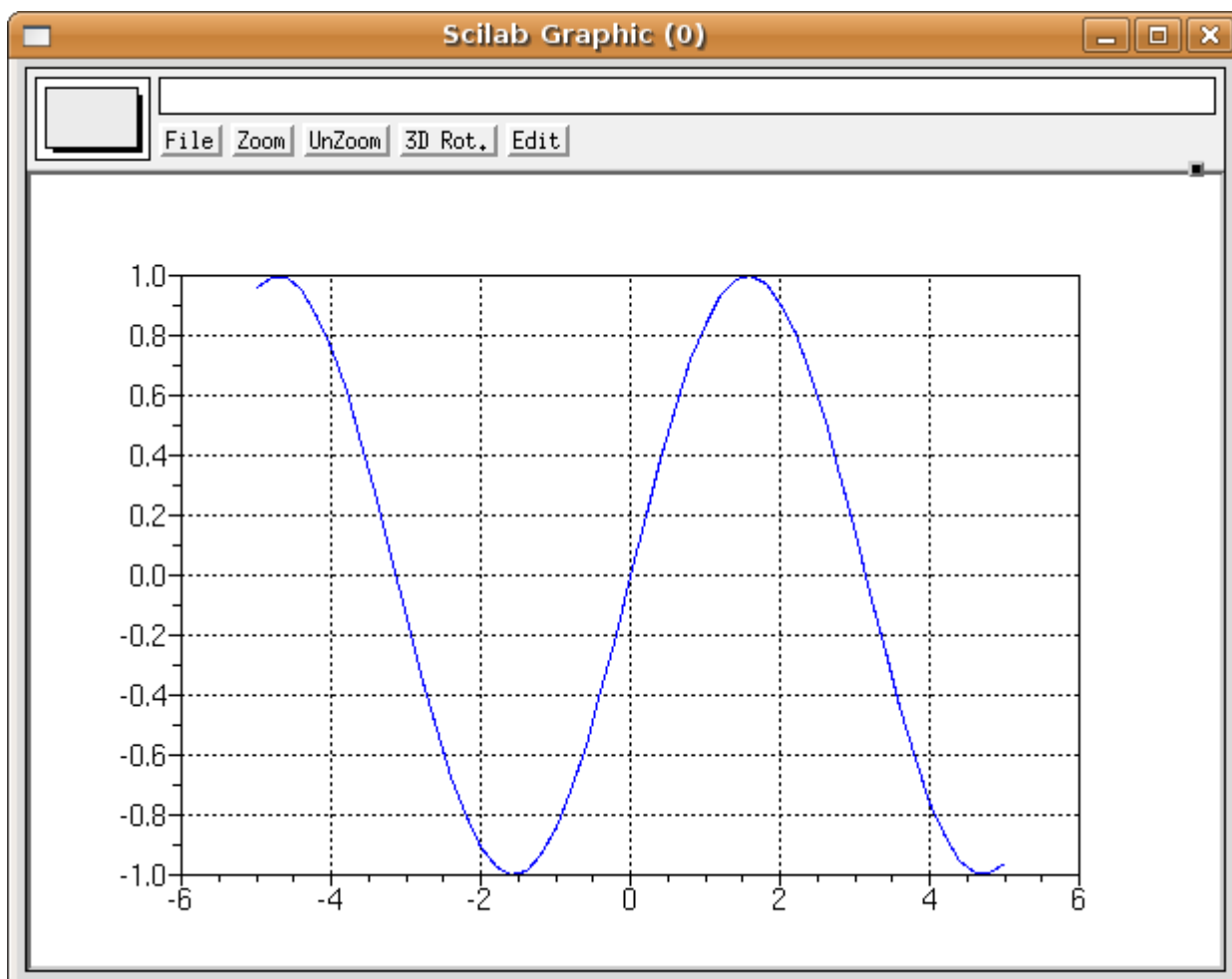


Рис. 10.7. График функции



Рис. 10.8. Окно с меткой

Одним из основных свойств метки является горизонтальное выравнивание, которое определяется свойством `HorizontalAlignment`. Это свойство может принимать одно из следующих значений:

- `left` – выравнивание по левому краю;
- `center` – выравнивание текста по центру (значение по умолчанию);
- `right` – выравнивание по правому краю.

В качестве примера рассмотрим окно, содержащее 4 текстовых поля с разными значениями свойства `HorizontalAlignment`. Текст программы представлен в листинге 10.7

```
hFig=figure();
set(hFig,'Position',[50,50,300,200]);
hSt1=uicontrol('Style','text','Position',[30,30,150,20],'String',
'Metka 1');
set(hSt1,'BackgroundColor',[1 1 1]);
set(hSt1,'HorizontalAlignment','left');
hSt2=uicontrol('Style','text','Position',[30,60,150,20],
'HorizontalAlignment','center','BackgroundColor',[1 1 1],'String',
'Metka 2');
hSt3=uicontrol('Style','text','Position',[30,90,150,20],'HorizontalA
lignment','right','BackgroundColor',[1 1 1],'String','Metka 3');
hSt4=uicontrol('Style','text','Position',[30,120,150,20],'Background
Color',[1 1 1],'String','Metka 4');
```

Листинг 10.7. Создание нескольких меток

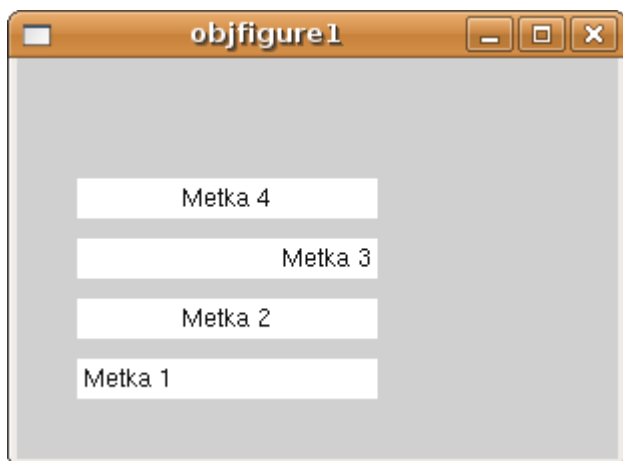


Рис. 10.9. Окно с несколькими метками

10.2.3. Компоненты Переключатель и Флажок

Рассмотрим еще два компонента – переключатель и флажок, которые позволяют переключаться между состояниями или выключать одно из свойств.

У флажка свойство 'Style' принимает значение 'checkbox', у переключателя свойство 'Style' – 'radiobutton'.

Индикатором альтернативных комбинаций является переключатель (Radiobutton), который также создается с помощью функции `uicontrol` (см. листинг 10.8 и рис. 10.10):

```
hFig=figure();
Rb=uicontrol('Style','radiobutton','String','name','value', 0,
'Position', [25,150,70,30]);
```

Листинг 10.8. Создание переключателя

При создании кнопки должно быть задано состояние кнопки (параметр 'value'), это может быть 1 (кнопка активна) или 0 (кнопка не активна). Задать значение свойства 'value' можно `printf` и с помощью функции `set`.

```
set(Rb,'value',0)
```

Переключатель, может реагировать на событие 'callback' и вызывать на выполнение определенную функцию.

Проиллюстрируем применение переключателей на примере выбора функции, график которой должен воспроизводиться в отдельном графическом окне (см. листинг 10.8 и рис. 10.11).

```
hFig=figure('Position',[50,50,200,200]);
//Создание радиокнопок
hRb1=uicontrol('Style','radiobutton','String','sin(x)','value',0,
'Position',[25,100,60,20],'callback','Radio');
```

```
hRb2=icontrol('Style','radiobutton','String','cos(x)','value',0,
'Position',[25,140,60,20],'callback','Radio');
```



Рис. 10.10. Окно с переключателем

```
//Функция Radio, реагирующая на событие 'callback'
function Radio()
newaxes;
x=-2*pi:0.1:2*pi;
if get(hRb1,'value')==1 //Если активна первая кнопка
set(hRb2,'value',0); //Сброс альтернативной кнопки
y=sin(x);
plot(x,y,'-r'); //Построение синусоиды
xgrid();
else
if get(hRb2,'value')==1
set(hRb1,'value',0);
y=cos(x);
plot(x,y,'-r'); //Построение косинусоиды
xgrid(); //Нанесение сетки на график
end;
end;
endfunction
```

Листинг 10.8. Пример работы с переключателями

Компонент флажок используется для индикации неальтернативных комбинаций. Генерация события `callback` и автоматическое выделение кнопки происходят при щелчке на квадратике или сопровождающей его надписи. Если флажок включен, то значение свойства `value` равно 1.

Щелчок по флажку автоматически изменяет состояние на противоположное. Использование флажка аналогично переключателю.

10.2.4. Компонент окно редактирования

Интерфейсный элемент окно редактирования (у того компонента свойство 'Style' должно принимать значение 'text') может использоваться для ввода и вывода символьной информации. Текст, набираемый в окне редактирования, можно корректировать, при работе с компонентом можно использовать операции с буфером обмена. Процедура ввода, завершаемая нажатием клавиши **Enter**, генерирует событие Callback.

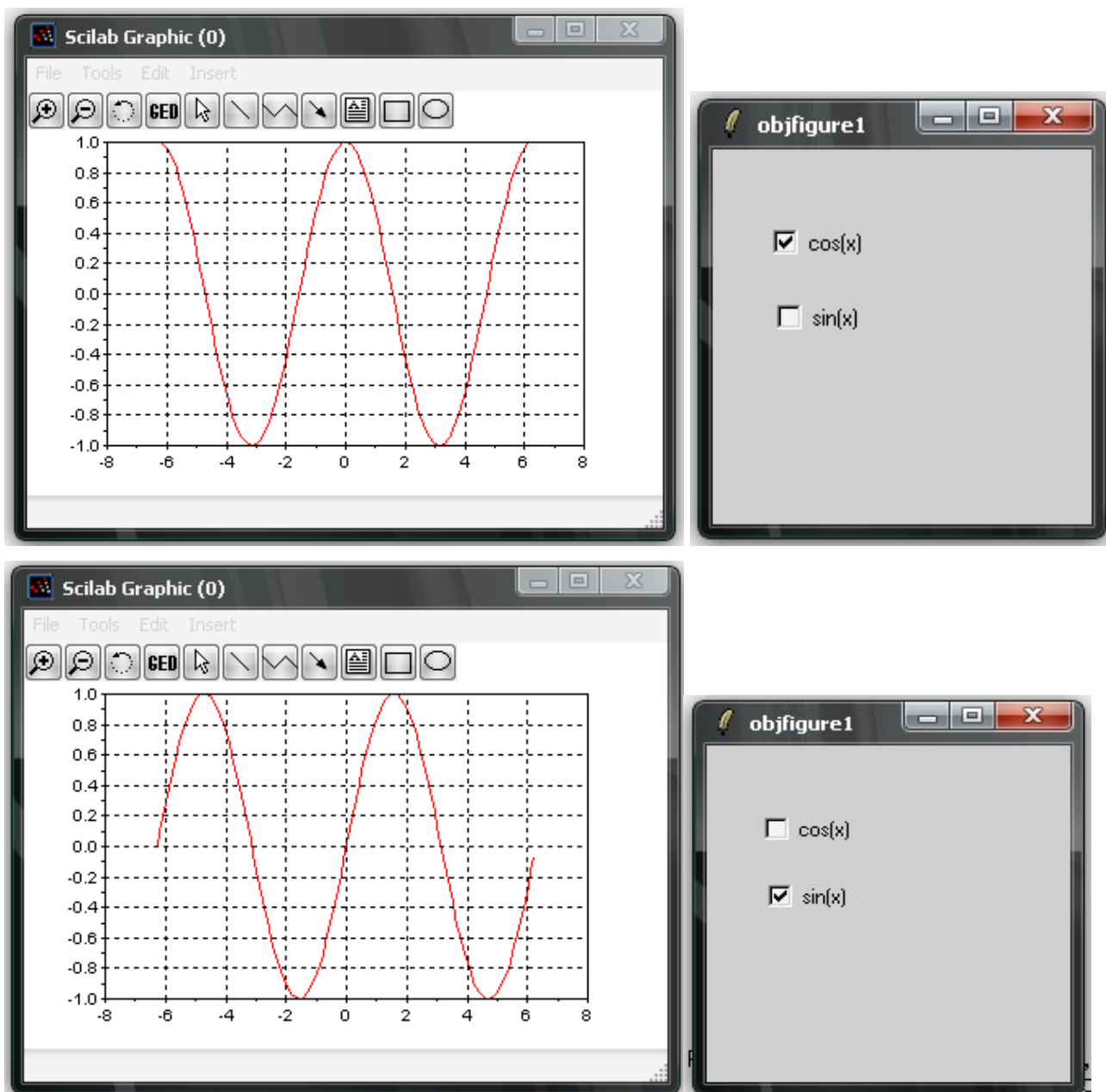


Рис. 10. 11. Окно с переключателем

Строка ввода определяется параметром 'String', которое определяет находящийся в компоненте текст. Для нормального функционирования компонента этот параметр необходимо обязательно задавать при определении компонента с помощью функции `uicontrol`. Изменить значение этого свойства можно с помощью функции `set`, а считать значение этого свойства можно с помощью функции `get`.

Вводимый текст может быть прижат к левому или правому краю окна ввода, если задать соответствующее значение свойства `HorizontalAlignment`. Если вводимый текст представляет собой числовое значение, которое должно быть использовано в работе программы, то содержимое свойства `string` переводится в числовой формат с помощью функции `evstr` (будет рассмотрено далее на примере квадратного уравнения).

В качестве примера рассмотрим работы с несколькими компонентами рассмотрим следующую задачу.

Написать программу решения квадратного или биквадратного уравнения. Выбор типа уравнения будем проводить с помощью компонента **Переключатель**. Программа представлена на листинге 10.9.

```
f=figure(); //Создание графического объекта
set(f,'position',[0,0,500,300])
set(f,'figure_name','EQUATION');
//Создание меток для полей ввода коэффициентов
lab_a=uicontrol(f,'style','text','string','A=','position',[50, 250,
100, 20]);
lab_b=uicontrol(f,'style','text','string','B=','position',[150, 250,
100, 20]);
lab_c=uicontrol(f,'style','text','string','C=','position',[250, 250,
100, 20]);
// полей ввода коэффициентов
edit_a=uicontrol(f,'style','edit','string','1','position',[50, 230,
100, 20]);
edit_b=uicontrol(f,'style','edit','string','2','position',[150, 230,
100, 20]);
edit_c=uicontrol(f,'style','edit','string','1','position',[250, 230,
100, 20]);
//Текстовое поле, определяющее вывод результатов
```

```

text_result=icontrol(f,'style','text','string','', 'position',[50,
80, 450, 20]);
//Флажок, отвечающая за выбор типа уравнения
radio_bikv=icontrol('style','radiobutton','string','Bikvadrat?',
'value',1,'position',[100,100,200,20]);
BtSolve=icontrol('style','pushbutton','string','Solve','Callback',
'Solve','position',[50,50,60,20]);
BtClose=icontrol('style','pushbutton','string','Close','Callback',
'_Close','position',[400,50,60,20]);
//Функция решения уравнения
function Solve()
// Считываем значение переменных из текстовых полей и
// преобразовываем их числовому типу
a=eval(get(edit_a,'string'));
b=eval(get(edit_b,'string'));
c=eval(get(edit_c,'string'));
d=b*b-4*a*c;
// Проверяем значение флажка
if get(radio_bikv,'value')==0
if d<0
set(text_result,'string','No Solve kvadrat');
else
x1=(-b+sqrt(d))/2/a;
x2=(-b-sqrt(d))/2/a;
set(text_result,'string',sprintf("2          kornya          kvadrat\t
x1=%1.2f\tx2=%1.2f",x1,x2));
end;
else
if d<0
set(text_result,'string','No Solve bikvadrat ');
else
y1=(-b+sqrt(d))/2/a;
y2=(-b-sqrt(d))/2/a;
if (y1<0) & (y2<0)

```

```

        set(text_result,'string','No Solve bikbadrat');
elseif (y1>=0)&(y2>=0)
    x1=sqrt(y1);x2=-x1;x3=sqrt(y2);x4=-x3;
    set(text_result,'string',sprintf("4 kornya bikvadrat \t
x1=%1.2f\tx2=%1.2f\tx3=%1.2f\tx4=%1.2f",x1,x2,x3,x4));
else
    if y1>=0
        x1=sqrt(y1);x2=-x1;
    else
        x1=sqrt(y2);x2=-x1;
    end;
    set(text_result,'string',
    sprintf(" 2 kornya bikvadrat \t x1=%1.2f\tx2=%1.2f",x1,x2));
end;
end;
end
endfunction
// Функция закрытия окна
function _Close()
close(f)
endfunction

```

Листинг 10.9. Решения квадратного или биквадратного уравнения

На рисунке 10.12 представлено окно программы.

10.2.5. Списки строк

Интерфейсный компонент «список строк» в простейшем случае можно рассматривать как окно с массивом строк в нем. Если длина списка превышает высоту окна, то для перемещения по списку может использоваться вертикальная полоса прокрутки, которая генерируется автоматически.

Создание списка строк производится с помощью функции `uicontrol`, при задании параметра `'Style' - 'listbox'`. Рассмотрим это на простом примере (см листинг 10.10 и рис. 10.13).

```

f=figure();
// создание графического окна

```

```
h=uicontrol(f,'style','listbox','position',[10 10 150 160]);  
// создание listbox  
set(h,'string','item 1|item 2|item3');  
// заполнение списка  
set(h,'value',[1 3]);  
// выделение item 1 и 3 в списке
```

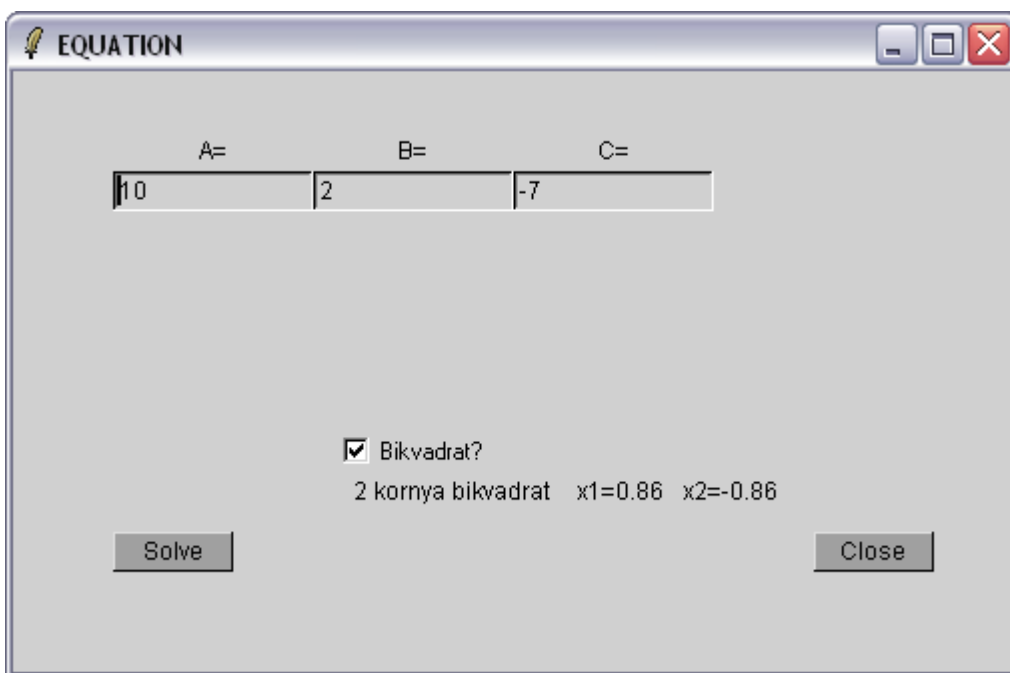
Листинг 10.10. Создание списка

Рис. 10.12. Окно программы решения уравнения



Рис. 10.13. Окно со списком

Список позволяет пользователю выбрать одну или несколько строк и в зависимости от выбора произвести то или иное действие.

Выбор строки осуществляется щелчком левой кнопки мыши в тот момент, когда острие курсора указывает на выбираемую строку. Одновременно с подсветкой строки ее номер заносится

в свойство `value` и генерируется событие `callback`. Строки в списке нумеруются от 1. Для выбора разрозненных строк нужно зажать клавишу **Ctrl** и щелкать мышью по выделяемым строкам. При этом каждая выделяемая строка подсвечивается, а ее номер запоминается в векторе `value`. Для выбора группы подряд идущих строк можно нажать и удерживать клавишу **Shift**, а затем щелкнуть по первой и последней строке группы. Все промежуточные строки тоже будут выделены и все их номера запомнятся в векторе `value`. В следующем параграфе будет приведено большое количество примеров использования этих (и других) компонентов при программировании различных задач.

11. Обработка экспериментальных данных

11.1. Метод наименьших квадратов

Метод наименьших квадратов позволяет по экспериментальным данным подобрать такую аналитическую функцию, которая проходит настолько близко к экспериментальным точкам, насколько это возможно.

Пусть в результате эксперимента были получены некоторые данные, отображенные в виде таблицы (табл. 11.1). Требуется построить аналитическую зависимость, наиболее точно описывающую результаты эксперимента.

Таблица 11.1. Экспериментальные данные

x_i	x_1	x_2	x_3	x_4	x_5	x_6	x_7	...	x_n
y_i	y_1	y_2	y_3	y_4	y_5	y_6	y_7	...	y_n

Идея метода наименьших квадратов заключается в том, что функцию

$$Y = f(x, a_0, a_1, \dots, a_k)$$

необходимо подобрать таким образом, чтобы сумма квадратов отклонений измеренных значений y_i от расчетных Y_i была наименьшей:

$$S = \sum_{i=1}^n (y_i - f(x_i, a_0, a_1, \dots, a_k))^2 \rightarrow \min \quad (11.1)$$

Задача сводится к определению коэффициентов a_i из условия (11.1). Для реализации этой задачи в Scilab предусмотрена функция $[a, S] = \text{datafit}(F, z, a_0)$,

где

F - функция параметра, которой необходимо подобрать;

z - матрица исходных данных;

c - вектор начальных приближений;

a - вектор коэффициентов;

S - сумма квадратов отклонений измеренных значений от расчетных.

Рассмотрим использование функции `datafit` на примере.

ЗАДАЧА 11.1. В результате опыта холостого хода определена зависимость потребляемой из сети мощности (P_0 , Вт) от входного напряжения (U_1 , В) для асинхронного двигателя.

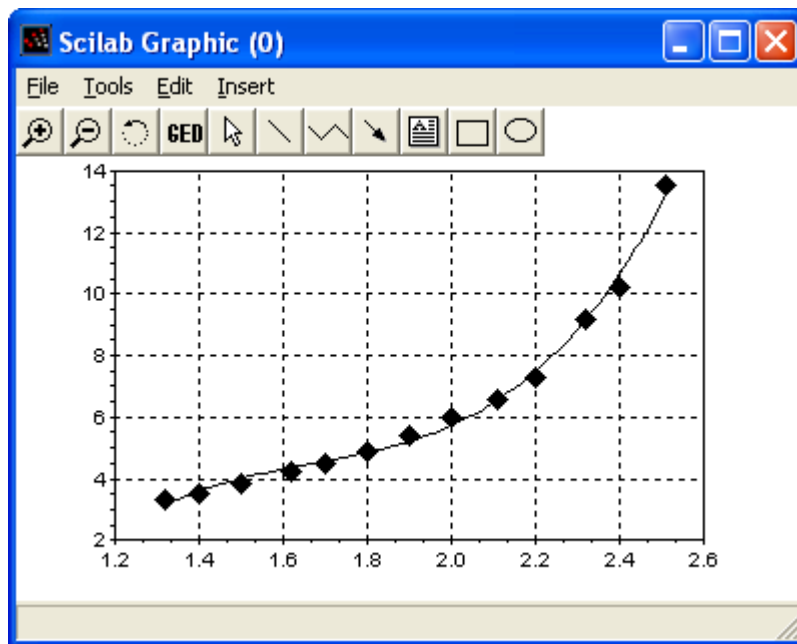
$U_1, \text{В}$	132	140	150	162	170	180	190	200	211	220	232	240	251
$P_0, \text{Вт}$	330	350	385	425	450	485	540	600	660	730	920	1020	1350

Методом наименьших квадратов подобрать зависимость вида

$$P_0 = c_1 + c_2 U_1 + c_3 U_1^2 + c_4 U_1^3 \quad (11.2).$$

Решение задачи показано в листинге 11.1. Геометрическая интерпретация на рис. 11.1.

```
//Функция (11.2)
function [zr]=G(c,z)
zr=z(2)-c(1)-c(2)*z(1)-c(3)*z(1)^2-c(4)*z(1)^3
endfunction
//Исходные данные
x=[1.32 1.40 1.50 1.62 1.70 1.80 1.90
    2.00 2.11 2.20 2.32 2.40 2.51];
y=[3.30 3.50 3.85 4.25 4.50 4.85 5.40
    6.00 6.60 7.30 9.20 10.20 13.50];
//Построение графика экспериментальных данных
plot2d(x,y,-4);
//Вектор начальных приближений
c=[0;0;0;0];
//Формирование матрицы исходных данных
z=[x;y];
//Решение задачи
[c,err]=datafit(G,z,c);
// Построение графика подобранной функции
t=1.32:0.01:2.51;
Ptc=c(1)+c(2)*t+c(3)*t^2+c(4)*t^3;
plot2d(t,Ptc);
```

Листинг 11.1*Рис. 11.1. Решение задачи 11.1*

11.2. Интерполяция функций

Простейшая задача *интерполирования* заключается в следующем. На отрезке $[a; b]$ заданы точки $x_0, x_1, x_2, \dots, x_n$ (всего $n + 1$ точка), которые называют *узлами интерполяции*, и значения некоторой функции $f(x)$ в этих точках:

$$f(x_0)=y_0, f(x_1)=y_1, f(x_2)=y_2, \dots, f(x_n)=y_n. \quad (11.2)$$

Требуется построить *интерполирующую функцию* $F(x)$, принадлежащую известному классу и принимающую в узлах интерполяции те же значения, что и $f(x)$:

$$F(x_0)=y_0, F(x_1)=y_1, F(x_2)=y_2, \dots, F(x_n)=y_n. \quad (11.3)$$

Для решения подобной задачи довольно часто используют сплайн-интерполяцию (от английского слова spline – рейка, линейка). Один из наиболее распространенных вариантов интерполяции – интерполяция кубическими сплайнами. Кроме того существуют квадратичные и линейные сплайны.

В Scilab для построения линейной интерполяции служит функция `y=interpLn(z, x)`, где

`z` - матрица исходных данных;

`x` - вектор абсцисс;

`y` - вектор значений линейного сплайна в точка `x`;

Пример использования функции `interpLn` показан в листинге 11.2. Здесь линейный сплайн применяется для решения задачи 11.1. Графическое решение задачи показано на рис. 11.2.

```
x=[132 140 150 162 170 180 190 200 211 220 232 240 251];
y=[330 350 385 425 450 485 540 600 660 730 920 1020 1350];
plot2d(x, y, -4);
z=[x; y];
t=132:5:252;
ptd=interpLn(z, t)
plot2d(t, ptd);
xgrid();
```

Листинг 11.2

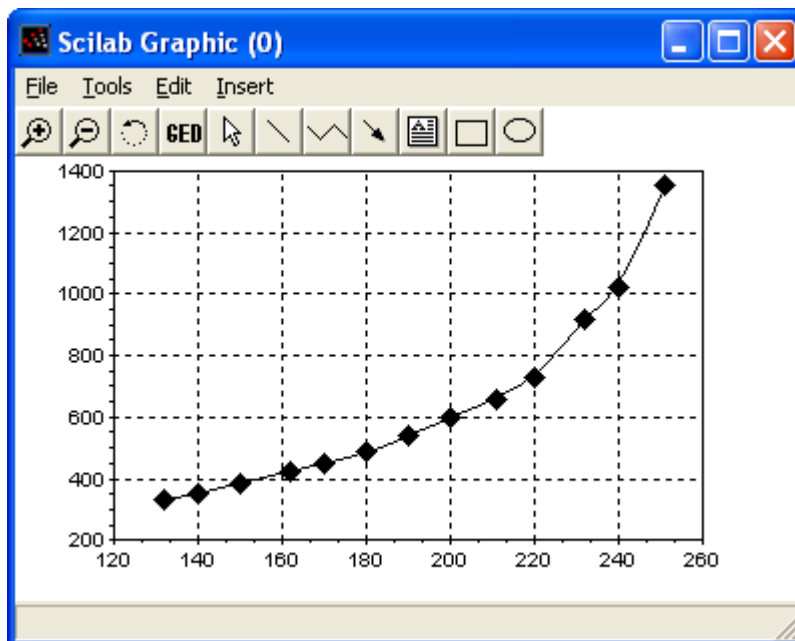


Рис. 11.2. Геометрическая интерпретация линейного сплайна

Построение кубического сплайна в Scilab состоит из двух этапов: вначале вычисляются коэффициенты сплайна с помощью функции `d=splin(x, y)`, а затем рассчитываются значения интерполяционного полинома в точке `y=interp(t, x, y, d)`.

Функция `d=splin(x, y)` имеет следующие параметры:

`x` – строго возрастающий вектор, состоящий минимум из двух компонент;

`y` – вектор того же формата, что и `x`;

`d` – результат работы функции, коэффициенты кубического сплайна.

Для функции `y=interp(t, x, y, k)` параметры `x`, `y` и `d` имеют те же значения, параметр `t` это вектор абсцисс, а `y` - вектор ординат, являющихся значениями кубического сплайна в точках `x`.

Листинг 11.3 содержит пример решения задачи 11.1 с помощью интерполяции кубическими сплайнами.

```
x=[132 140 150 162 170 180 190 200 211 220 232 240 251];  
y=[330 350 385 425 450 485 540 600 660 730 920 1020 1350];  
plot2d(x, y, -4);  
coeff=splin(x, y);  
t=132:5:252;  
ptd=interp(t, x, y, coeff);  
plot2d(t, ptd);  
xgrid();
```

Листинг 11.3

12 Решение дифференциальных уравнений в частных производных

Математические модели физических и иных процессов описываются с помощью дифференциальных уравнений в частных производных¹. Аргументами функций этих уравнений являются пространственные координаты x , y , z и время t . Общие сведения о дифференциальных уравнениях в частных производных приведены в первом параграфе главы. В Scilab, как и в большинстве математических пакетов, нет средств для непосредственного решения уравнений математической физики. Однако, возможностей пакета достаточно, для реализации метода сеток решения дифференциальных уравнений в частных производных. В последующих параграфах этой главы и описана реализация метода сеток для решения параболических, гиперболических и эллиптических уравнений в Scilab.

12.1 Общие сведения о дифференциальных уравнениях в частных производных

Линейным уравнением в частных производных второго порядка называется соотношение между функцией $u(x,y)$ (или $u(x,t)$) и ее частными производными вида [1]:

$$L(u) = A(x, y) \frac{\partial^2 u}{\partial x^2} + 2B(x, y) \frac{\partial^2 u}{\partial x \partial y} + C(x, y) \frac{\partial^2 u}{\partial y^2} + D(x, y) \frac{\partial u}{\partial x} + E(x, y) \frac{\partial u}{\partial y} + G(x, y)u(x, y) = F(x, y) \quad (12.1)$$

Если переменная функция u зависит от x и t , то уравнение (12.1) может быть записано следующим образом:

$$L(u) = A(x, t) \frac{\partial^2 u}{\partial x^2} + 2B(x, t) \frac{\partial^2 u}{\partial x \partial t} + C(x, t) \frac{\partial^2 u}{\partial t^2} + D(x, t) \frac{\partial u}{\partial x} + E(x, t) \frac{\partial u}{\partial t} + G(x, t)u(x, t) = F(x, t) \quad (12.2)$$

В случае если в правой части уравнения $F=0$, то уравнения (12.1)-(12.2) называются однородными, иначе неоднородными [2].

Если $B^2 - 4AC < 0$, то уравнение (12.2) относится к классу *эллиптических уравнений*, если $B^2 - 4AC > 0$, то (12.2) это - *гиперболическое уравнение*, $B^2 - 4AC = 0$ если - *параболическое уравнение*. В случае, когда $B^2 - 4AC$ не имеет постоянного знака, то это уравнение смешанного типа.

С помощью преобразования переменных x , y (или x , t) уравнение можно привести к виду, когда $B=0$. В этом случае очень просто определяется тип уравнения. Если A и C имеют один и тот же знак, то уравнение (12.2) *эллиптическое уравнение*, если разные, то *гиперболическое*, а если A или C равно 0, то уравнение относится к *параболическим* [2].

К классическим эллиптическим уравнениям относятся [1][2]:

- уравнение *Лапласа* $\Delta u = 0$ ², которое используется для описания магнитных и

¹ В литературе эти уравнения часто называют уравнениями математической физики

² В двумерном случае оператор Лапласа имеет вид $\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$, в трехмерном

$$\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} /$$

стационарных тепловых полей.

- уравнение Пуассона $\Delta u = f$, которое применяется в электростатике, теории упругости и т. д.
- уравнение Гельмгольца $\Delta u + cu = f$, описывающее установившиеся колебательные процессы.

Среди гиперболических уравнений можно выделить [3]:

- волновые уравнения: одномерное волновое уравнение $\frac{\partial^2 u}{\partial t^2} = a^2 \frac{\partial^2 u}{\partial x^2} + f(x, t)$, которое описывает вынужденные колебания струны; двумерное волновое уравнение $\frac{\partial^2 u}{\partial t^2} = a^2 \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) + f(x, y, t)$ описывает колебания мембраны³.
- телеграфное уравнение $\frac{\partial^2 u}{\partial t^2} + \frac{RC + LG}{LC} \frac{\partial u}{\partial t} + \frac{RG}{LC} u - \frac{1}{LC} \frac{\partial^2 u}{\partial x^2} = 0$ описывает изменение потенциала u в линиях электропередачи, L, C, R, G коэффициент самоиндукции, емкость, сопротивление, характеристика потерь на единицу длины линии.

К классическим параболическим уравнениям относится уравнение теплопроводности $\frac{\partial u}{\partial t} = a^2 \Delta u + f$.

Для нахождения единственного решения дифференциального уравнения в частных производных необходимо задать начальные и граничные условия. Начальными условиями принято называть условия, заданные в начальный момент времени t . Граничные условия задаются при различных значениях пространственных переменных. Для эллиптических уравнений задаются только граничные условия, которые можно разделить на три класса[3]:

- условие Дирихле $u_{(x, y, z) \in \Gamma} = \varphi(x, y, z)$, в этом случае на границе области Γ , в которой ищется решение, задана некая непрерывная функция φ . В одномерном случае это условие принимает вид $u(0, t) = \varphi_1(t), u(L, t) = \varphi_2(t)$, $(0, L)$ - интервал, на котором ищется решение одномерной задачи;
- условие Неймана $\frac{\partial u}{\partial n}_{(x, y, z) \in \Gamma} = \varphi(x, y, z)$, в этом случае на границе области задана производная по направлению n внешней нормали.
- смешанное условие $\left(\alpha u + \beta \frac{\partial u}{\partial n} \right)_{(x, y, z) \in \Gamma} = \varphi(x, y, z)$.

Для параболических уравнений, кроме граничных условий, необходимо определить одно начальное, которое может быть таким: $u(x, t_0) = \psi(x)$.

В случае гиперболических уравнений начальные условия могут быть следующими:

$$u(x, t_0) = \psi_1(x) \quad \text{и} \quad \frac{\partial u(x, t_0)}{\partial t} = \psi_2(x).$$

Существуют аналитические методы решения уравнений в частных производных, такие как метод Фурье (метод разделения переменных), в результате применения которых решение записывается в виде суммы бесконечного ряда довольно сложной структуры, и нахождение численного значения функции в конкретной точке представляет собой отдельную математическую задачу. Поэтому широкое распространение получили численные методы

³ При $f=0$ уравнение описывает свободные колебания струны или мембраны.

решения уравнений в частных производных.

12.2 Использование метода сеток для решения параболических уравнений в частных производных

Одним из наиболее распространенных численных методов решения уравнений является *метод сеток*⁴ [1]. В методе сеток область Ω , в которой необходимо найти решение уравнения, прямыми, параллельными осям $t=t_j$ и $x=x_i$, разобьем на прямоугольные области (см. рис. 7.1), где $x_i = x_0 + ih, h = \frac{x_n - x_0}{n}, i = 0, 1, 2, \dots, n,$

$t_j = t_0 + j\Delta, \Delta = \frac{t_k - t_0}{k}, j = 0, 1, \dots, k$. Точки, которые лежат на границе Γ области Ω , называются *внешними*, остальные точки *внутренними*. Совокупность всех точек называется сеткой, Ω_h^Δ величины h и Δ шагами сетки по x и t соответственно.

Идея метода сеток состоит в том, что вместо любой непрерывной функции $w(x, t)$ будем рассматривать дискретную функцию $w_i^j = w(x_i, t_j)$, которая определена в узлах сетки Ω_h^Δ , вместо производных функции будем рассматривать их простейшие разностные аппроксимации в узлах сетки. Таким образом, вместо системы дифференциальных уравнений в частных производных получим систему алгебраических уравнений. Чем меньше величины h и Δ , тем точнее получаемые алгебраические уравнения моделируют исходное дифференциальное уравнение в частных производных. В этом и последующих параграфах этой главы будет рассмотрен метод сеток для каждого из трех типов уравнений и его реализация в Scilab. Знакомство с численными методами решения дифференциальных уравнений в частных производных начнем с разностных схем решения параболических уравнений.

Разностные схемы решения параболических уравнений будем рассматривать на примере следующего одномерного уравнения (12.3):

$$\begin{aligned} \frac{\partial u}{\partial t} &= a^2 \frac{\partial^2 u}{\partial x^2} + f(x, t), 0 \leq x \leq L, 0 \leq t \leq T \\ u(0, t) &= \mu(t), u(L, t) = \eta(t), 0 \leq t \leq T \\ u(x, 0) &= \varphi(x), 0 \leq x \leq L \end{aligned} \quad (12.3)$$

⁴ Метод сеток в литературе также называют методом конечных разностей.

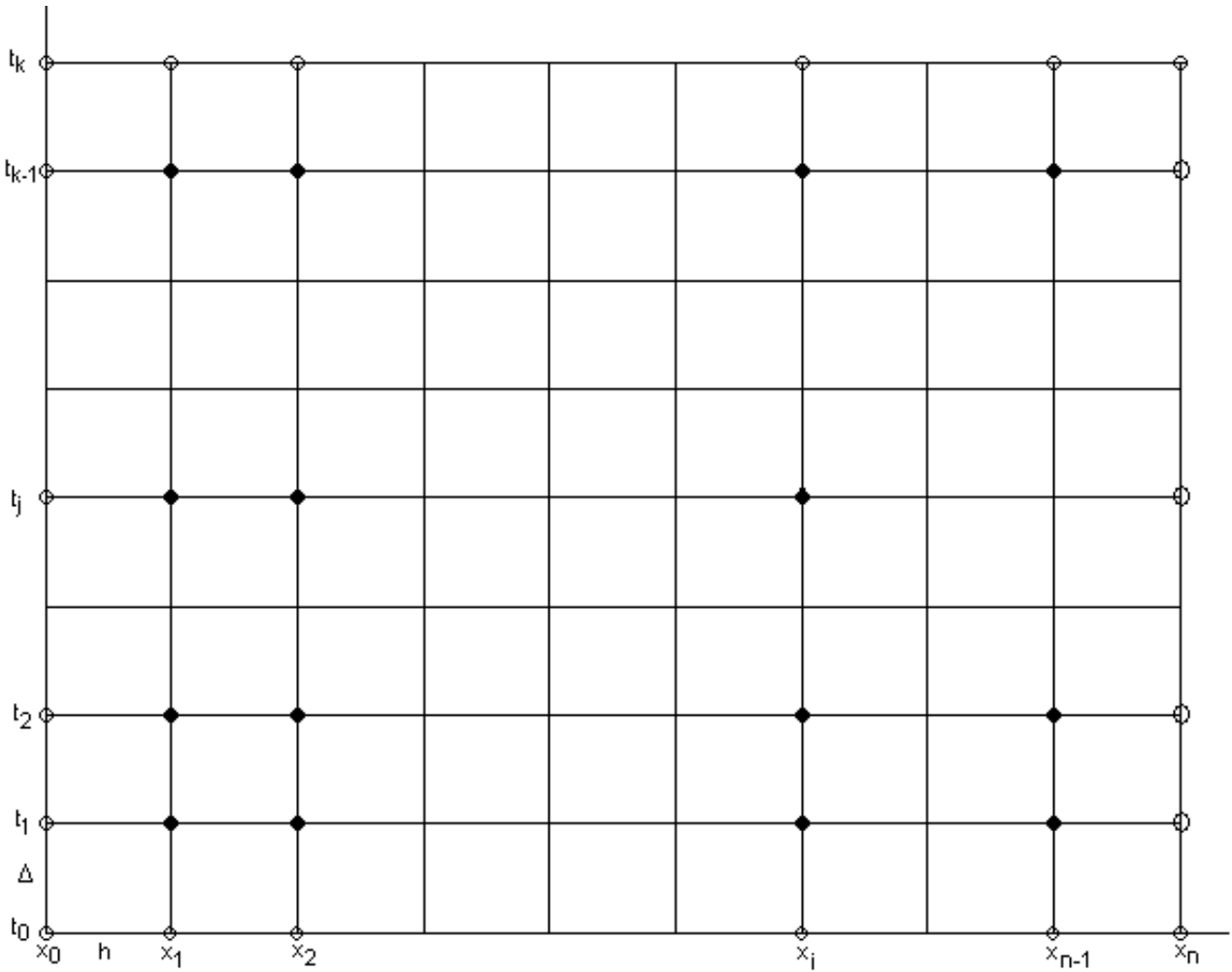


Рисунок 12.1. Сетка Ω_h^Δ для области Ω с границей Γ

Построим сетку Ω_h^Δ (см. рис. 12.1). Для получения сеточного уравнения заменим производную $\frac{\partial^2 u}{\partial x^2}$ приближенной разностной формулой[1]:

$$\frac{\partial^2 u(x_i, t_j)}{\partial x^2} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} . \quad (12.4)$$

В этой и последующих формулах $u_{i,j}$ - это значение функции u в точке (x_i, t_j) , $u_{i+1,j}$ в точке (x_{i+1}, t_j) , $u_{i-1,j}$ в точке (x_{i-1}, t_j) , $u_{i,j+1}$ в точке (x_i, t_{j+1}) и $u_{i,j-1}$ в точке (x_i, t_{j-1}) .

Для замены $\frac{\partial u}{\partial t}$ можно воспользоваться одной из приближенных разностных формул [1]

$$\frac{\partial u(x_i, t_j)}{\partial t} = \frac{u_{i,j+1} - u_{i,j}}{\Delta} \quad (12.5)$$

$$\frac{\partial u(x_i, t_j)}{\partial t} = \frac{u_{i,j} - u_{i,j-1}}{\Delta} \quad (12.6)$$

Кроме того, заменим начальные и граничные условия их разностной аппроксимацией:

$$u_{i,0} = \varphi(x_i) = \varphi_i, \quad i = 0, 1, \dots, n \quad (12.7)$$

$$u_{0,j} = \mu(t_j) = \mu_j, \quad u_{n,j} = \nu(t_j) = \nu_j, \quad j = 0, 1, \dots, k \quad (12.8)$$

Заменяя частные производные в задаче (12.3) соотношениями (12.4) и (12.5) и учитывая условия (12.7)-(12.8), получим следующую вычислительную схему для расчета значений функции u в узлах сетки Ω_h^Δ

$$u_{i,j+1} = \gamma u_{i,j-1} + (1 - 2\gamma) u_{i,j} + \gamma u_{i,j+1} + \Delta f_{i,j} \quad (12.9)$$

$$u_{0,j} = \mu_j, \quad u_{n,j} = \nu_j, \quad u_{i,0} = \varphi_i, \quad \gamma = \frac{a^2 \Delta}{h^2} \quad (12.10)$$

Это явная двухслойная разностная схема (см. рис. 7.2).

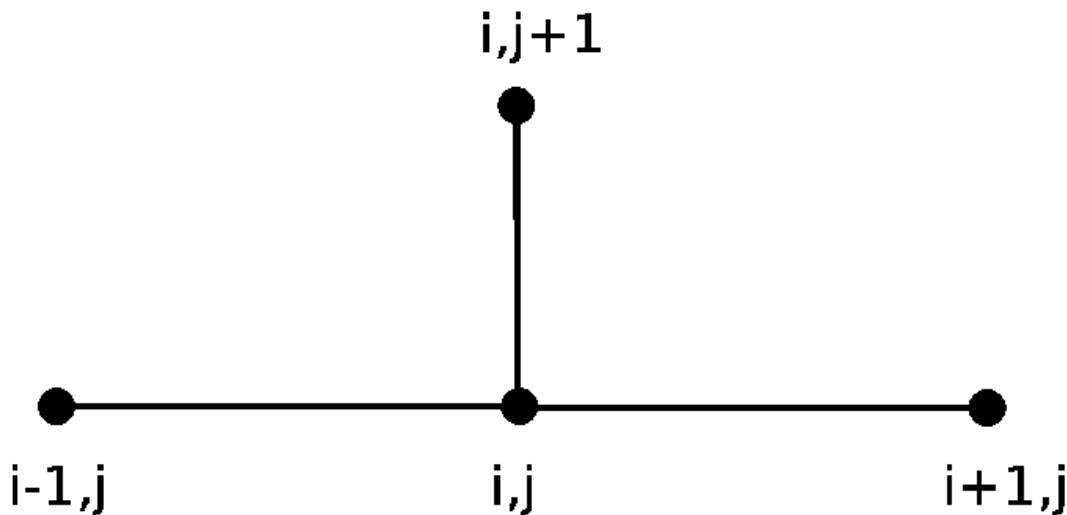


Рисунок 12.2. Шаблон явной двухслойной разностной схемы

Учитывая, что на нулевом слое (при $i=0$) все значения $u_{i,0}$, (как впрочем, и $u_{0,j}$, $u_{n,j}$) известны, по формуле (12.9) можно сначала явно рассчитать значения $u_{i,1}$, затем $u_{i,2}$ и так до $u_{i,k}$. Для устойчивости разностной схемы (12.9) значения шагов по t и x должны удовлетворять следующему условию

$$\Delta \leq \frac{h^2}{2a^2} \quad (12.11)$$

Рассмотрим решение параболического уравнения на примере следующей задачи.

ЗАДАЧА 12.1.

Решить параболическое уравнение, описывающее распределение температуры в стержне длиной L , начальная температура стержня задается произвольной функцией $\varphi(x)$. Температуры концов стержня равны $u(0, t) = U_1 = const$, $u(L, t) = U_2 = const$.

$$\frac{\partial u}{\partial t} = a^2 \frac{\partial^2 u}{\partial x^2} + f(x, t), \quad a^2 = \frac{\lambda}{c\rho}, \quad 0 < x < L, \quad 0 < t < \infty, \\ u(0, t) = U_1 = \text{const}, \quad u(L, t) = U_2 = \text{const}, \quad 0 < t < \infty, \\ u(x, 0) = \varphi(x), \quad 0 < x < L$$
(12.12)

здесь a^2 коэффициент температуропроводности, λ коэффициент теплопроводности материала стержня, c удельная теплоемкость, ρ плотность массы.

Подпрограмма решения задачи 12.1 с помощью явной разностной схемы (12.9) - (12.10) в Scilab представлена на листинге 12.1⁵.

```
//Правая часть дифференциального уравнения.
function y=f(x,t)
y=sin(x*t)
endfunction
//Начальное условие
function y=fi(x)
y=exp(0.15*x)
endfunction
//Условие на левой границе
function y=му(t)
y=1
endfunction
//Условие на правой границе
function y=ну(x)
y=2.117
endfunction
function [u,x,t]=parabol(N,K,L,T,a)
//Функция решения параболического уравнения методом сеток с
//помощью явной разностной схемы. N - количество участков,
// на которые разбивается интервал по x (0,L); K – количество
// участков, на которые разбивается интервал по t (0,T); a -
// параметр дифференциального уравнения теплопроводности,
// Функция возвращает матрицу решений u и вектора x, t
// Вычисляем шаг по x
h=L/N;
// Вычисляем шаг по t
delta=T/K;
// Формируем массив x и первый столбец матрицы решений U
// из начального условия
for i=1:N+1
x(i)=(i-1)*h;
u(i,1)=fi(x(i));
end
//Формируем массив t, первую и последнюю строку матрицы
решений
// U из граничных условий
for j=1:K+1
```

⁵ В листинге 12.1 и во всех последующих уже учтен тот факт, что массивы в Scilab нумеруются с 0.

```

t(j)=(j-1)*delta;
u(1,j)=myu(t(j));
u(N+1,j)=nyu(t(j));
end
gam=a^2*delta/h^2;
// Формируем матрицу решений u с помощью явной разностной
схемы
// (12.9)
for j=1:K
for i=2:N
u(i,j+1)=gam*u(i-1,j)+(1-2*gam)*u(i,j)+gam*u(i+1,j)+delta*...
f(x(i),t(j));
end
end
end

```

Листинг 12.1. Подпрограмма решения задачи 12.1 с помощью явной разностной схемы

Входными данными подпрограммы *parabol* решения задачи 12.1 являются: N - количество интервалов, на которые разбивается отрезок $(0, L)$; K - количество интервалов, на которые разбивается отрезок $(0, T)$; L - длина стержня, T - интервал времени, a - параметр дифференциального уравнения. Функция возвращает три параметра: решение - сеточная функция u , определенная на сетке Ω_h^Δ , массивы x и t .

На листинге 12.2 представлено обращение к функции *parabol* для решения задачи 12.1 и построения графика решения, который изображен на рис. 12.3.

```

[U,X,T]=parabol(50,200,5,3,0.4);
surf(X,T,U');
title('PARABOLIC EQUATION');
xlabel('X');
ylabel('T');

```

Листинг 12.2. Вызов функции parabol

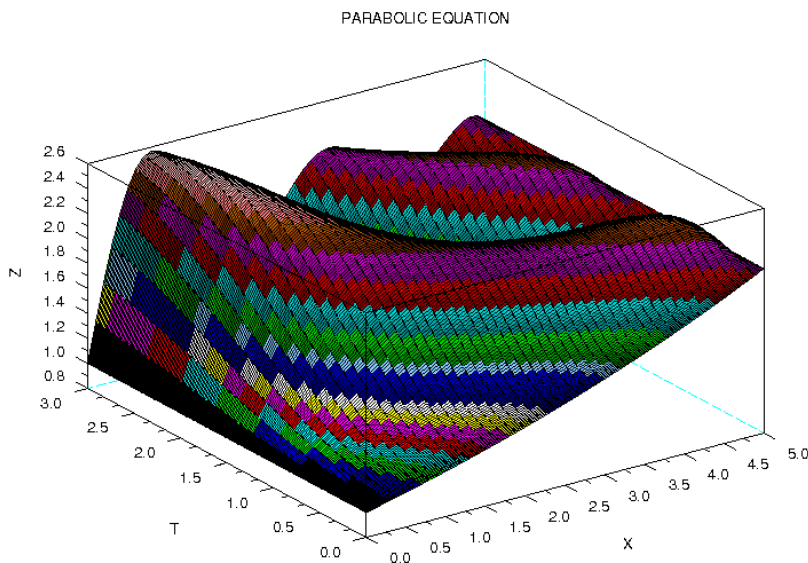


Рисунок 12.3. График решения задачи 12.1 при $f(x,t)=\sin(xt)$

При решении параболических уравнений с помощью явной разностной схемы основной проблемой является устойчивость решения и правильный выбор шага по t , удовлетворяющего соотношению (12.11).

Для решения этой проблемы были предложены неявные разностные схемы[1]. Эти схемы абсолютно устойчивы, но алгоритм решения получаемого сеточного уравнения несколько сложнее, чем простой пересчет по формуле (12.9). Рассмотрим неявную разностную схему для параболического уравнения. Для построения неявной разностной схемы заменим частные производные в задаче (12.3) соотношениями (12.4), (12.6⁶) и с учетом условий (12.7)-(12.8) получим следующую вычислительную схему для расчета значений функции u в узлах сетки Ω_h^Δ .

$$\begin{aligned} \gamma u_{i-1,j} - (1+2\gamma)u_{i,j} + \gamma u_{i+1,j} &= -u_{i,j-1} - \Delta f_{i,j}, \\ i=1,2,\dots,n-1, j=1,2,\dots,k \end{aligned} \quad (12.13)$$

Соотношения (12.13) вместе с равенствами (12.10) - *неявная двухслойная* разностная схема (см. рис. 12.4). Схема (12.10), (12.13) не позволяет явно выписать решение, и для нахождения $u_{i,j}$ при каждом значении j необходимо решить трехдиагональную систему линейных алгебраических уравнений, для чего можно воспользоваться одним из итерационных методов (например методом Зейделя [1]) или методом прогонки[1]. Преобразуем систему (12.13) к следующему виду:

$$u_{i,j} = \frac{\gamma}{1+2\gamma} (u_{i-1,j} + u_{i+1,j}) + \frac{u_{i,j-1}}{1+2\gamma} + \frac{\Delta}{1+2\gamma} f(x_i, t_j) \quad (12.14)$$

Формула (12.14) позволит запрограммировать решение системы, получаемой с помощью неявной разностной схемы, одним из численных методов, например, с помощью метода Зейделя.

Решение параболического уравнения с помощью неявной разностной схемы (с помощью функции *neuvnp*) представлено на листинге 12.3.

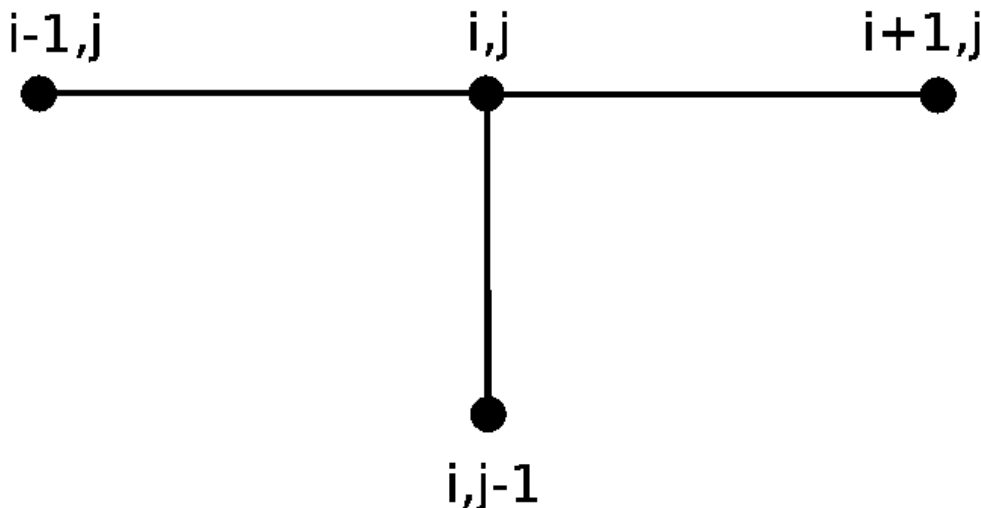


Рисунок 12.4. Шаблон неявной двухслойной разностной схемы

Входными данными функции *neuvnp* являются: N - количество участков, на которые разбивается интервал по x ($0, L$); K - количество участков, на которые разбивается интервал по t ($0, T$); a - параметр дифференциального уравнения теплопроводности; eps - точность решения СЛАУ (12.4) методом Зейделя⁷. Функция *neuvnp* возвращает матрицу решений u

⁶ Вместо (12.5) для явной схемы

⁷ Выбор метод Зейделя при программировании трехдиагональной систем линейных алгебраических

задачи 12.1; массивы x и t ; r - точность решения системы (12.4) методом Зейделя; количество итераций k .

```
//Правая часть дифференциального уравнения.
function y=f(x,t)
y=sin(x*t)
//y=0;
endfunction
//Начальное условие
function y=fi(x)
y=exp(0.15*x)
endfunction
//Условие на левой границе
function y=myu(t)
y=1
endfunction
//Условие на правой границе
function y=nyu(x)
y=2.117
endfunction
function [u,x,t,r,k]=neyavn(N,K,L,T,a,eps)
// Функция решения параболического уравнения методом сеток с
// помощью неявной разностной схемы. N - количество участков,
// на которые разбивается интервал по x (0,L);
// K - количество участков, на которые разбивается интервал по
// t (0,T); a - параметр дифференциального уравнения
// теплопроводности, eps - точность решения СЛАУ (12.14)
// методом Зейделя.
// Функция neyavn возвращает:
// u - матрицу решений в узлах сетки, массив x, массив t,
// r - точность решения системы (12.14) методом Зейделя, k -
// количество итераций при решении системы (7.14) методом
// Зейделя.
// Вычисляем шаг по x
h=L/N;
// Вычисляем шаг по t
delta=T/K;
// Формируем массив x и первый столбец матрицы решений U
// из начального условия
for i=1:N+1
x(i)=(i-1)*h;
u(i,1)=fi(x(i));
end
// Формируем массив t, первую и последнюю строку матрицы
// решений U из граничных условий
for j=1:K+1
t(j)=(j-1)*delta;
u(1,j)=myu(t(j));
```

уравнений вида (12.13) - (12.14) - субъективный выбор авторов. Читатель может использовать и другой итерационный метод или специализированный метод решения подобных систем, типа метода прогонки.

```

u(N+1,j)=nyu(t(j));
end
// Определяем матрицу ошибок R и заполняем ее нулями
R(N+1,K+1)=0;
// Вычисляем коэффициент gamma
gam=a^2*delta/h^2;
r=1;
k=0;
// Цикл while для организации итерационного процесса при
// решении системы уравнений (12.14) методом Зейделя с
// точностью eps
while r>eps
// Вычисление матрицы ошибок R во внутренних точках
// и пересчет значений u во внутренних точках при решении
СЛАУ
// (12.14) методом Зейделя
for i=2:N
for j=2:K+1
R(i,j)=abs(u(i,j)-gam/(1+2*gam)*(u(i-1,j)+u(i+1,j))-
u(i,j-1)...
/(1+2*gam)-delta*f(x(i),t(j))/(1+2*gam));
u(i,j)=gam/(1+2*gam)*(u(i-1,j)+u(i+1,j))+u(i,j-1)...
/(1+2*gam)+delta*f(x(i),t(j))/(1+2*gam);
end
end
// Поиск максимума в матрице ошибок
r=R(1,1);
for i=1:N+1
for j=1:K+1
if R(i,j)>r
r=R(i,j);
end
end
end
// Увеличение количества итерации.
k=k+1;
end
disp(k)
endfunction
[U,X,T]=neyavn(50,200,5,3,0.4,0.1);
surf(X,T,U');
title('PARABOLIC EQUATION');
xlabel('X');
ylabel('T');

```

Листинг 12.3. Решение задачи 12.1 с помощью неявной разностной схемы⁸

Графики решений с помощью явной и неявной разностных схем практически

⁸ Читатель может самостоятельно сравнить результаты, полученные с помощью явной и неявной разностных схем.

совпадают.

Обратите внимание, что функции, приведенные на листингах 12.1 и 12.3, позволяют решать уравнения вида (12.3) с различными функциями $f(x, t)$, $\mu(t)$, $\eta(t)$, $\varphi(x)$.

Использование неявной разностной схемы в случае, когда $f(x, t) \neq 0$, рассмотрим на примере еще одной задачи.

ЗАДАЧА 12.2.

Найти распределение температуры в стержне длиной L , начальная температура стержня задается произвольной функцией $f(x)$. Температуры концов стержня равны $u(0, t) = U_1 = \text{const}$, $u(L, t) = U_2 = \text{const}$. На боковой поверхности стержня происходит теплообмен по закону Ньютона со средой, температура которой равна u_0 .

Начально-граничная задача, описывающая распределение температуры стержня, имеет вид:

$$\begin{aligned} \frac{\partial u}{\partial t} &= a^2 \frac{\partial^2 u}{\partial x^2} - h(u - u_0), \quad a^2 = \frac{\lambda}{c\rho}, \quad h = \frac{\alpha p}{c\rho\sigma}, \quad 0 < x < L, \quad 0 < t < \infty, \\ u(0, t) &= U_1 = \text{const}, \quad u(L, t) = U_2 = \text{const}, \quad 0 < t < \infty, \\ u(x, 0) &= \varphi(x), \quad 0 < x < L \end{aligned} \quad (12.15)$$

здесь α коэффициент теплообмена, σ площадь поперечного сечения стержня, p периметр поперечного сечения стержня.

Построим сетку Ω_h^Δ $(x_i = ih, h = \frac{L}{n}, i = 0, 1, 2, \dots, n)$,

$(t_j = j\Delta, \Delta = \frac{T}{k}, j = 0, 1, \dots, k)$. Для получения сеточного уравнения заменим производную

$\frac{\partial^2 u}{\partial x^2}$ и $\frac{\partial u}{\partial t}$ приближенными разностными формулами (12.4), (12.6). Получится

следующая неявная разностная схема (12.16)-(12.18):

$$\begin{aligned} u_{i,0} &= \varphi(x_i), \quad i = 0, 1, \dots, N, \\ u_{0,j} &= U_1, \quad u_{N,j} = U_2, \quad j = 0, 1, \dots, K \end{aligned} \quad (12.16)$$

$$u_{i,j} = \frac{1}{1 + 2\gamma + \Delta h} u_{i,j-1} + \frac{\gamma}{1 + 2\gamma + \Delta h} (u_{i-1,j} + u_{i+1,j}) + \frac{\Delta h}{1 + 2\gamma + \Delta h} u_0, \quad (12.17)$$

$i = 1, 2, \dots, N-1; j = 1, 2, \dots, k;$

$$\gamma = a^2 \frac{\Delta}{hx^2} \quad (12.18)$$

Применение неявной разностной схемы для решения задачи 12.2 представлено на листинге 12.4.

```
//Начальное условие
function y=fi(x)
y=exp(0.15*x)
endfunction
function [u,x,t,r,k]=neiav(N,K,L,T,a,h,U1,U2,u0,eps)
//Функция решения параболического уравнения методом сеток с
// помощью неявной разностной схемы.
//N - количество участков, на которые разбивается интервал по
```

x

```

// (0,L); K - количество участков, на которые разбивается
// интервал по t (0,T);
//a, h - параметры дифференциального уравнения
теплопроводности;
// eps - точность решения СЛАУ (7.17) методом Зейделя;
// U1 - температура на левом конце стержня;
// U2 - температура на правом конце стержня;
// Функция peiav возвращает:
// u - матрицу решений в узлах сетки, массив x, массив t,
// r - точность решения системы (12.17) методом Зейделя,
// k - количество итераций при решении системы методом
Зейделя.
// Вычисляем шаг по x
hx=L/N;
// Вычисляем шаг по t
delta=T/K;
// Формируем массив x и первый столбец матрицы решений U
// из начального условия
for i=1:N+1
x(i)=(i-1)*hx;
u(i,1)=fi(x(i));
end
// Формируем массив t, первую и последнюю строку матрицы
// решений U из граничных условий
for j=1:K+1
t(j)=(j-1)*delta;
u(1,j)=U1;
u(N+1,j)=U2;
end
// Определяем матрицу ошибок R и заполняем ее нулями
R(N+1,K+1)=0;
// Вычисляем коэффициент gamma
gam=a^2*delta/hx^2;
r=1;
k=0;
//Цикл while для организации итерационного процесса при
решении
//системы уравнений (12.17) методом Зейделя с точностью eps
while r>eps
// Вычисление матрицы ошибок R во внутренних точках
//и пересчет значений u во внутренних точках при решении СЛАУ
// (12.17) методом Зейделя
for j=2:K+1
for i=2:N
V=gam*(u(i-1,j)+u(i+1,j))/(1+2*gam+delta*hx)+u(i,j-1)/...
(1+2*gam+delta*hx)+delta*h*u0/(1+2*gam+delta*hx);
R(i,j)=abs(V-u(i,j));
u(i,j)=V;
end

```



```

end
// Поиск максимума в матрице ошибок
r=R(1,1);
for i=1:N+1
for j=1:K+1
if R(i,j)>r
r=R(i,j);
end
end
end
// Увеличение количества итераций
k=k+1;
end
endfunction
//Вызов функции решения задачи 12.2.
[U,X,T,R,K]=neiav(50,200,5,3,0.4,0.5,1,2.117,30,0.001);
//Построение графика функции
surf(X,T,U');
title('Example 12.2');
xlabel('X');
ylabel('T');

```

Листинг 12.4. Функция `neiav` решения задачи 12.2 с помощью неявной разностной схемы

Входные и выходные данные функции `neiav` решения задачи 12.2 описаны в комментариях листинга 12.4. На рис. 12.5 представлены результаты решения задачи.

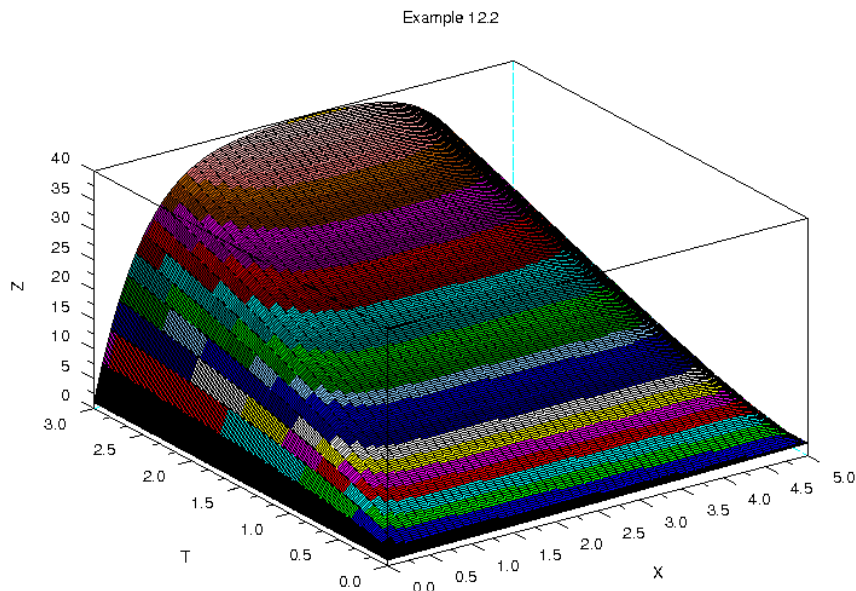


Рисунок 12.5. График решения задачи 12.2

Для решения получаемого алгебраического уравнения методом Зейделя потребовалось 1079 итераций. Поэтому для уменьшения количества итераций имеет смысл попробовать ускорить итерационный процесс с помощью методов релаксации или градиентных методов решения систем алгебраических уравнений Методика, изложенная в этом параграфе, может

быть использована и при решении других параболических уравнений.

12.3 Использование метода сеток для решения гиперболических уравнений

Решение гиперболических уравнений также можно осуществить с помощью разностных схем. Разностные схемы решения одномерного гиперболического уравнения рассмотрим на примере следующего уравнения:

$$\begin{aligned} \frac{\partial^2 u}{\partial t^2} &= a^2 \frac{\partial^2 u}{\partial x^2} + f(x, t), 0 < x < L, 0 \leq t \leq T \\ u(0, t) &= \mu(t), u(L, t) = \eta(t), 0 \leq t \leq T \\ u(x, 0) &= \varphi(x), \frac{\partial u(x, 0)}{\partial t} = \psi(x), 0 \leq x \leq L \end{aligned} \quad (12.19)$$

Построим сетку Ω_h^Δ (см. рис. 12.1), в которой будем искать решение уравнения (12.19). Частную производную $\frac{\partial^2 u}{\partial x^2}$ заменим разностным соотношением (12.4), а производную $\frac{\partial^2 u}{\partial t^2}$ - соотношением (12.20) [1].

$$\frac{\partial^2 u(x_i, t_j)}{\partial t^2} = \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{\Delta^2} \quad (12.20)$$

Подставляя (12.20), (12.4), (12.5) в гранично-начальную задачу (12.19), получим следующую явную разностную схему решения уравнения:

$$\begin{aligned} u_{i,j+1} &= -u_{i,j-1} + \gamma(u_{i-1,j} + u_{i+1,j}) = (2 - 2\gamma)u_{i,j} + \Delta^2 f_{i,j} \\ i &= 1, 2, \dots, N-1, j = 1, 2, \dots, K-1 \\ u_{i,0} &= \varphi(x_i), \frac{u_{i,1} - u_{i,0}}{\Delta} = \Psi_i, i = 0, 1, \dots, N \\ u_{0,j} &= \mu_j, u_{N,j} = \nu_j, j = 0, 1, \dots, K \\ \gamma &= \frac{a^2 \Delta^2}{h^2} \end{aligned} \quad (12.21)$$

которая устойчива при $\gamma < 1$ и по аналогии с разностной схемой (12.9)-(12.10) может быть легко запрограммирована в Scilab.

В качестве примера рассмотрим следующую начально-граничную задачу.

ЗАДАЧА 12.3.

Решить начально-граничную задачу

$$\begin{aligned} \frac{\partial^2 \omega}{\partial t^2} &= a^2 \frac{\partial^2 \omega}{\partial x^2} + \sin(xt), 0 < x < L, t > 0, \\ \omega(0, t) &= \varphi(0), \omega(L, t) = \varphi(L) \\ \omega(x, 0) &= \varphi(x), \omega_t(x, 0) = \psi(x) \end{aligned} \quad (12.22)$$

На листинге 12.5 представлена функция ggg решения уравнения (12.22), а на рис. 7.9 - график полученного решения. Параметры функции ggg аналогичны рассмотренным ранее подпрограммам решения параболических уравнений.

function [u,x,t]=ggg(N,K,L,T,a)

```

//Функция решения гиперболического уравнения с помощью явной
// разностной схемы. Входные данные:
// N - количество участков, на которые разбивается интервал по
// x (0,L); K - количество участков, на которые разбивается
// интервал по t (0,T); a - параметр дифференциального
// уравнения теплопроводности. Выходные данные:
//u - матрица решений в узлах сетки, массив x, массив t,
// Вычисляем шаг по x
h=L/N;
// Вычисляем шаг по t
delta=T/K;
//Формируем массив x, первый и второй столбцы матрицы решений
u
//из начального условия
for i=1:N+1
x(i)=(i-1)*h;
u(i,1)=fi(x(i));
u(i,2)=u(i,1)+delta*psi(x(i));
end
//Формируем массив t, первую и последнюю строку матрицы
решений
// U из граничных условий
for j=1:K+1
t(j)=(j-1)*delta;
end
//Формируем первую и последнюю строку матрицы решений U
//из граничных условий
for j=2:K+1
u(1,j)=0;
u(N+1,j)=fi(L);
end
gam=a^2*delta^2/h^2;
//Формируем матрицу решений u с помощью явной разностной схемы
//(12.22)
for j=2:K
for i=2:N
u(i,j+1)=-u(i,j-1)+gam*u(i-1,j)+(2-2*gam)*...
u(i,j)+gam*u(i+1,j)+delta^2*f(x(i),t(j));
end
end
end

```

Листинг 12.5. Функция ggg решения задачи 12.3 с помощью явной разностной схемы

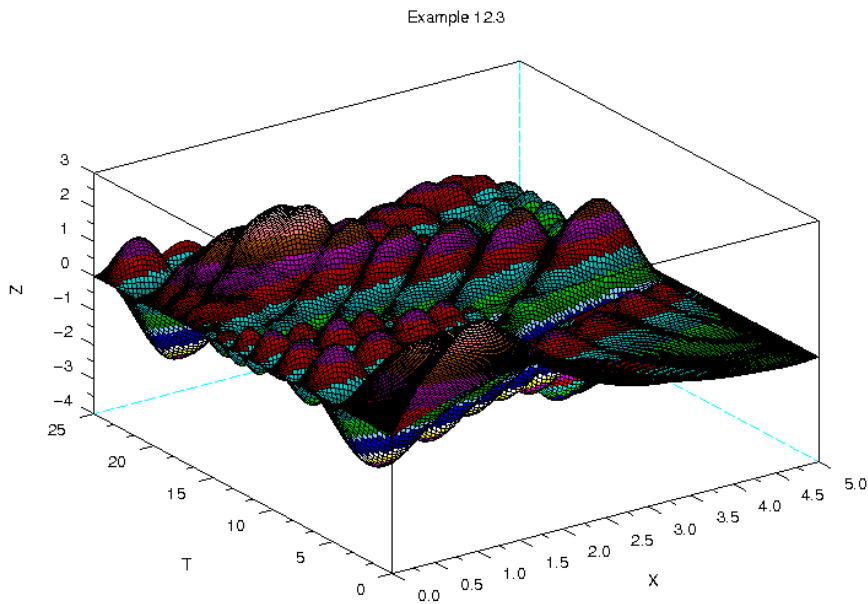


Рисунок 12.6. График решения задачи 12.3

Для решения гиперболических уравнений можно построить и неявные схемы, однако в связи с тем, что у них нет таких преимуществ перед явными, как в параболических уравнениях, их имеет смысл использовать только когда нельзя построить явную (например, при смешанных условиях на границе области).

12.4 Использование метода сеток для решения эллиптических уравнений

Рассмотрим разностную схему для эллиптического уравнения в прямоугольной области $\Omega(R-b \leq x \leq R+b, -a \leq y \leq a)$ с граничными условиями Дирихле на границе Γ .

ЗАДАЧА 12.4

$$\Delta u = \frac{\partial^2 \Psi}{\partial x^2} + \frac{\partial^2 \Psi}{\partial y^2} - \frac{5}{x} \frac{\partial \Psi}{\partial x} = -2$$

$$\Psi_{(x,y) \in \Gamma} = 0$$
(12.23)

Построим сетку Ω_{hx}^{hy} , для чего проведем в области Ω прямые, параллельные осям $y=y_j$ и $x=x_i$, где $x_i = R-b + i \cdot hx, hx = \frac{2b}{n}, i=0,1,2,\dots,Nx$,

$y_j = -a + j \cdot hy, hy = \frac{2a}{k}, j=0,1,\dots,Ny$. Для построения разностного уравнения заменим частные производные и граничные условия следующими соотношениями:

$$\frac{\partial^2 \Psi(x_i, y_j)}{\partial x^2} = \frac{\Psi_{i-1,j} - 2\Psi_{i,j} + \Psi_{i+1,j}}{hx^2}$$

$$\frac{\partial^2 \Psi(x_i, y_j)}{\partial y^2} = \frac{\Psi_{i,j-1} - 2\Psi_{i,j} + \Psi_{i,j+1}}{hy^2}$$
(12.24)

$$\begin{aligned}\Psi_{i,0}=\Psi_{i,Ny}=0, \quad i=0,1,\dots,Nx \\ \Psi_{0,j}=\Psi_{Nx,j}=0, \quad j=0,1,\dots,Ny\end{aligned}\quad (12.25)$$

С помощью соотношений (12.24)-(12.25) преобразуем эллиптическую краевую задачу к следующей системе разностных уравнений.

$$\begin{aligned}\Psi_{i,j}=\frac{1}{A}\left(B_i\Psi_{i+1,j}+C_i\Psi_{i-1,j}+D(\Psi_{i,j-1}+\Psi_{i,j+1})+2\right) \\ A=\frac{2}{hx^2}+\frac{2}{hy^2}, B_i=\frac{1}{hx^2}+\frac{5}{2hx x_i}, C_i=\frac{1}{hx^2}-\frac{5}{2hx x_i}, D=\frac{1}{hy^2} \\ i=1,2,\dots,Nx-1; j=1,2,\dots,Ny-1 \\ \Psi_{i,0}=\Psi_{i,Ny}=0, i=0,1,\dots,Nx \\ \Psi_{0,j}=\Psi_{Nx,j}=0, i=0,1,\dots,Ny\end{aligned}\quad (12.26)$$

Эту систему можно решать итерационными методами (например методом Зейделя). В случае медленной сходимости итерационных процессов при решении сеточных уравнений, получаемых при аппроксимации гиперболических и эллиптических задач, имеет смысл попробовать заменить метод Зейделя градиентными методами (или методами релаксации). На листинге 12.6 представлено решение уравнения 12.23 сеточным методом, а на рис. 7.10 - график найденного решения.

```
function [psi,x,y,k]=ellip(R,a,b,Nx,Ny,eps)
// Функция ellip решения задачи 12.4.
// Входные данные:
// R, a, b - значения, определяющие область решения задачи,
//Nx - количество участков, на которые разбивается интервал по
// x(R-b,R+b);
// Ny - количество участков, на которые разбивается интервал
по
// y (-a,a);
// eps - точность решения уравнения (12.26) метоом Зейделя.
// Выходные данные:
// psi - матрица решений в узлах сетки, массив x, массив y,
//k - количество итерация при решении разностного уравнения
// (12.26) методом Зейделя.
// Вычисляем шаг по y
hy=2*a/Ny;
// Вычисляем шаг по x
hx=2*b/Nx;
// Формируем массив x, первый и последний столбцы матрицы
// решений psi из граничного условия
for i=1:Nx+1
x(i)=R-b+(i-1)*hx;
psi(i,1)=0;
psi(i,Ny+1)=0;
end;
// Формируем массив y, первую и последнюю строки матрицы
// решений psi из граничного условия
for j=1:Ny+1
y(j)=-a+(j-1)*hy;
psi(1,j)=0;
```

```

psi(Nx+1,1)=0;
end;
// Вычисляем коэффициенты разностного уравнения (12.26)
A=2/hy^2+2/hx^2;
D=1/hy^2;
for i=2:Nx+1
B(i)=1/hx^2+5/(2*hx*x(i));
C(i)=1/hx^2-5/(2*hx*x(i));
end
//Решение разностного уравнения (12.26) методом Зейделя с
// точностью eps
p=1;
k=0;
while p>eps
for i=2:Nx
for j=2:Ny
V=1/A*(B(i)*psi(i-1,j)+C(i)*psi(i+1,j)+D*(psi(i,j-1)...
+psi(i,j+1))+2);
R(i,j)=abs(V-psi(i,j));
psi(i,j)=V;
end
end
p=R(2,2);
for i=2:Nx
for j=2:Ny
if R(i,j)>p
p=R(i,j);
end
end
end
k=k+1;
end
endfunction
//Вызов функции решения задачи 12.4.
[PSI,X,Y,K]=ellip(18,3,6,32,16,0.01);
//Построение графика функции
surf(X,Y,PSI');
title('Example 12.4');
xlabel('X');
ylabel('Y');

```

Листинг 12.6. Решение задачи 12.4

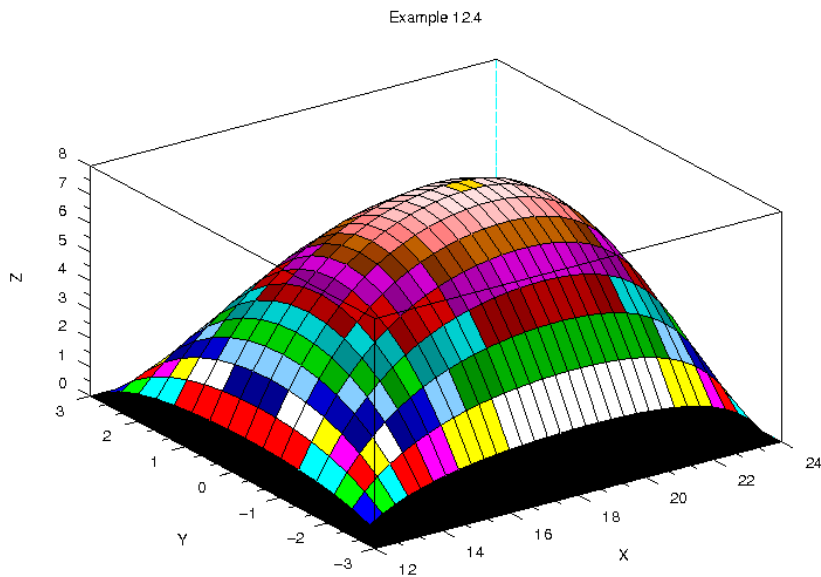


Рисунок 12.7. График решения уравнения 12.23 сеточным методом

Авторы надеются, что читатель, разобравшийся с решением уравнения 12.23, без проблем построит разностную схему и для других эллиптических уравнений.

Метод сеток позволяет решить широкий класс уравнений в частных производных. Однако при сложной геометрии области, уравнениях с переменными коэффициентами, сложными условиями на границе области использование этого метода не целесообразно, в этих случаях можно использовать метод конечных элементов, который реализован в кроссплатформенном свободно распространяемом пакете *freefem* [4].

13 Решение задач оптимизации

В этой главе будут рассмотрены возможности средства пакета Scilab для решения оптимизационных задач.

13.1 Поиск минимума функции одной переменной

В качестве простейшей оптимизационной задачи рассмотрим поиск локального минимума функции одной переменной.

ЗАДАЧА 13.1.

Найти минимум функции $f(x) = x^4 + 3x^3 - 13x^2 - 6x + 26$.

Решение задачи начнем с построения графика функции (см. листинг 13.1 и рис. 13.1).

```
x=-5:0.1:1;
```

```
y=x.^4+3*x.^3-13*x.^2-6*x+26;
```

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

```
xtitle('График функции f(x)=x^4+3*x^3-13*x^2-6*x+26','X','Y');
```

```
xgrid();
```

Листинг 13.1. Построение графика функции

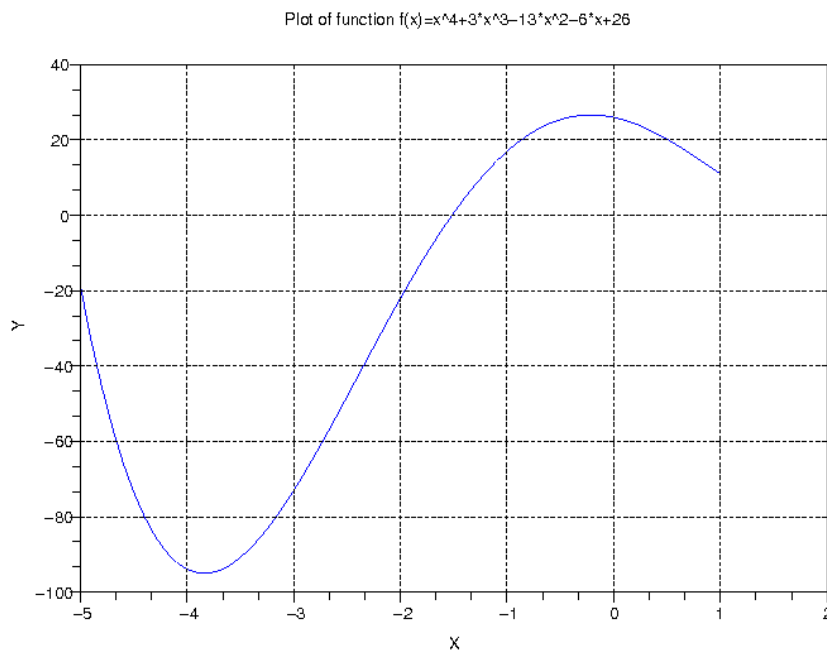


Рисунок 13.1. График функции $f(x) = x^4 + 3x^3 - 13x^2 - 6x + 26$

Построив график функции видно, что функция имеет минимум в районе точки -4. Для нахождения более точного значения минимума функции в Scilab служит функция

```
[f, xopt]=optim(costf, x0),
```

которая предназначена для поиска минимума любой функции, $x0$ — вектор столбец начальных приближений длиной n , функция *constf* определяет функцию, минимум которой ищется.

Функция возвращает минимум функции (f) и точку в которой функция достигает этого значения ($xopt$).

Главной особенностью функции *optim* является структура функции *costf*, которая должна быть следующей:


```
function [f,g,ind]=costf(x,ind)
//Функция costf должна возвращать функцию f, ее градиент g.
//f - функция от вектора неизвестных x, минимум которой ищется
f=gg(x);
// g - градиент функции f (вектор частной производной f по x),
g=numdiff(gg,x);1
endfunction
```

Для функции одной переменной в качестве f возвращается функция минимум которой ищется, в качестве функции g ее производная.

Если возвращаемое сформированной функцией *costf* значение *ind* равно 2, 3 или 4, то функция *costf* обеспечивает поиск минимума, т.е. в качестве результата функции *optim* возвращается f и $xopt$. Если $ind=1$, то в функции *optim* ничего не считается, условие $ind<0$ означает, что минимум $f(x)$ не может быть оценен, а $ind=0$ прерывает оптимизацию.

Вообще говоря, значение параметра *ind* является внутренним параметром для связи между *optim* и *costf*, для использования *optim* необходимо помнить, что параметр *ind* должен быть определен в функции *costf*.

Таким образом при использовании функции *optim* необходимо сформировать функцию *costf*, которая должна возвращать минимизируемую функцию f и ее градиент (производную).

Рассмотрим использование функции *optim* для поиска минимума $f(x)=x^4+3x^3-13x^2-6x+26$. Как видно из графика (см. рис. 13.1) минимум функция достигает в районе $x_{min} \approx 4$. На листинге представлено использование *optim* для поиска минимума функции одной переменной на примере $f(x)=x^4+3x^3-13x^2-6x+26$.

```
//Функция fi, в которой будет формироваться функция f и ее
// производная g.
function [f,g,ind]=fi(x,ind)
//Функция f, минимум которой ищется.
f=x^4+3*x^3-13*x^2-6*x+26
//Функция g - производная от функции f.
g=4*x^3+9*x^2-26*x-6
endfunction
// Начальное приближение точки минимума.
y0=-2;
//Для поиска точки минимума (xmin) и значения функции (fmin) в
// ней - вызов функции optim.
[fmin,xmin]=optim(fi,y0);
```

Листинг 13.2. Поиск минимума функции одной переменной.

Ниже представлены представлены полученные результаты поиска минимума

```
-->fmin
fmin =
- 95.089413
-->xmin
xmin =
- 3.8407084
```

Аналогично можно найти минимум любой другой функции одной переменной, главной задачей является проблема правильного выбора точки начального приближения. Но это проблема не пакета Scilab, а математическая проблема.

¹ Функция numdiff описана в седьмой главе.

13.2 Поиск минимума функции многих переменных

При нахождении минимума функции многих переменных функцию *costf* необходимо построить таким образом, чтобы входными данными в нее были значения вектора неизвестных x и параметра *ind*. Функция *costf* должна зависеть не от нескольких неизвестных, а от одного массива (вектора) неизвестных.

В случае функции многих переменных структура функции *costf* должна быть такой

```
function [f,g,ind]=costf(x,ind)
//f - функция от вектора неизвестных x, минимум которой ищется
f=gg(x);
// g - градиент функции f (вектор частной производной),
g=numdiff(gg,x);
endfunction
```

В качестве примера рассмотрим поиск минимума функции Розенброка $f(x,y)=100(y-x^2)^2+(1-x^2)^2$. График функции Розенброка представлен на рис. 13.2.

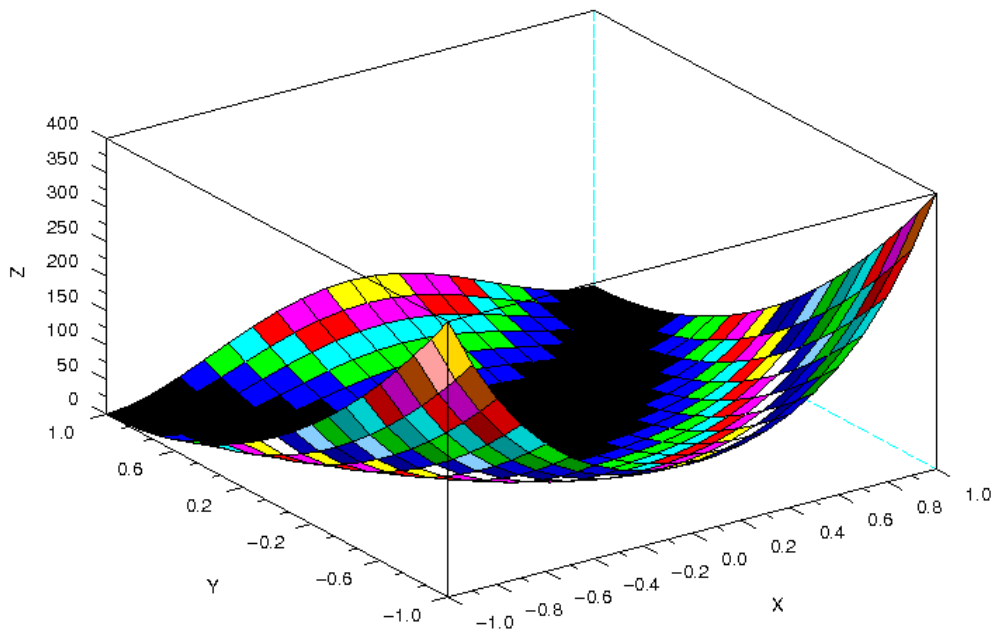


Рисунок 13.2. График функции Розенброка

Как известно функция Розенброка имеет минимум в точке (1,1) равный 0. В виду своей специфики функция Розенброка является тестовой для алгоритмов минимизации. Найдем минимум этой функции с помощью функции *optim* (листинг 13.2).

```
// Начальное приближение x0
x0=[-2;2]
//Функция Розенброка
function y=gg(x)
//Обратите внимание, здесь x - массив из двух неизвестных.
y=100*(x(2)-x(1)^2)^2+(1-x(1))^2;
endfunction
//Формирование функции cst, возвращающей функцию Розенброка и
//ее градиент.
function [f,g,ind]=cst(x,ind)
```

```
f=gg(x);
g=numdiff(gg,x);
endfunction
// Вызов функции optim
[f,xopt]=optim(cst,x0)
```

Листинг 13.3.

Ниже представлены результат поиска минимума функции Розенброка с помощью функции `optim`.

```
x0 =
-2.
 2.
xopt =
 0.9999955
 0.9999910
f =
 2.010D-11
```

13.3 Решение задач линейного программирования

Еще одной часто встречающихся в практике оптимизационных задач является задача линейного программирования. Знакомство с задачами линейного программирования начнем на примере задачи об оптимальном рационе.

Задача об оптимальном рационе. Имеется четыре вида продуктов питания: *П1*, *П2*, *П3*, *П4*. Известна стоимость единицы каждого продукта c_1, c_2, c_3, c_4 . Из этих продуктов необходимо составить пищевой рацион, который должен содержать не менее b_1 единиц белков, не менее b_2 единиц углеводов, не менее b_3 единиц жиров. Причем известно, в единице продукта *П1* содержится a_{11} единиц белков, a_{12} единиц углеводов и a_{13} единиц жиров и т.д. (см. таблицу 13.1).

Таблица 13.1. Содержимое белков, углеводов и жиров в продуктах

Элемент	белки	углеводы	жиры
<i>П1</i>	a_{11}	a_{12}	a_{13}
<i>П2</i>	a_{21}	a_{22}	a_{23}
<i>П3</i>	a_{31}	a_{32}	a_{33}
<i>П4</i>	a_{41}	a_{42}	a_{43}

Требуется составить пищевой рацион, чтобы обеспечить заданные условия при минимальной стоимости.

Пусть x_1, x_2, x_3, x_4 - количества продуктов *П1*, *П2*, *П3*, *П4*. Общая стоимость рациона равна

$$L = c_1 x_1 + c_2 x_2 + c_3 x_3 + c_4 x_4 = \sum_{i=1}^4 c_i x_i \quad (13.1)$$

Сформулируем ограничение на количество белков, углеводов и жиров в виде неравенств. В одной единице продукта *П1* содержится a_{11} единиц белков, в x_1 единицах - $a_{11}x_1$, в x_2 единицах продукта *П2* содержится $a_{21}x_2$ единиц белка и т.д.

Следовательно общее количество белков во всех четырех типов продукта равно $\sum_{j=1}^4 a_{jl}x_j$ и должно быть не больше b_1 . Получаем первое ограничение

$$a_{11}x_1 + a_{21}x_2 + a_{31}x_3 + a_{41}x_4 \leq b_1 \quad (13.2)$$

Аналогичные ограничения для жиров и углеводов имеют вид:

$$a_{12}x_1 + a_{22}x_2 + a_{32}x_3 + a_{42}x_4 \leq b_2 \quad (13.3)$$

$$a_{13}x_1 + a_{23}x_2 + a_{33}x_3 + a_{43}x_4 \leq b_3 \quad (13.4)$$

Принимаем во внимание, что x_1, x_2, x_3, x_4 положительные значения, получим еще четыре ограничения

$$x_1 \geq 0, x_2 \geq 0, x_3 \geq 0, x_4 \geq 0 \quad (13.5)$$

Таким образом задачу о рациональном рационе можно сформулировать следующим образом: найти значения переменных x_1, x_2, x_3, x_4 , удовлетворяющие системе ограничений (13.2) - (13.5), при которых линейная функция (13.1) принимала бы минимальное значение.

Задача об оптимальном рационе является задачей линейного программирования, функция (13.1) называется функцией цели, а ограничения (13.2) - (13.5) системой ограничений задачи линейного программирования.

В задачах линейного программирования функция цели функция цели L и система ограничений являются линейными.

В общем случае задачу линейного программирования можно сформулировать следующим образом. Найти такие значения x_1, x_2, \dots, x_n , удовлетворяющие системе ограничений (), при которых функция цели L (13.7) достигает своего минимального (максимального) значения

$$\sum_{j=1}^n a_{ij}x_j \leq b_i, i=1, \dots, m, \quad (13.6)$$

$$x_i \geq 0$$

$$L = c_1x_1 + c_2x_2 + \dots + c_nx_n = \sum_{i=1}^n c_i x_i \quad (13.7)$$

Для решения задач линейного программирования в Scilab предназначена функция *linpro* следующей структуры

$$[x, kl, f] = \text{linpro}(c, A, b[, ci, cs][, k][, x0])$$

Здесь c - массив (вектор-столбец) коэффициентов при неизвестных функции цели, длина вектора n совпадает с количеством неизвестных x , A - матрица при неизвестных из левой части системы ограничений, количество строк матрицы равно количеству ограничений m , а количество столбцов совпадает с количеством неизвестных n , b - массив (вектор-столбец) содержит свободные члены системы ограничений, длина вектора m , ci - массив (вектор-столбец) размерностью n содержит нижнюю границу переменных $(ci_j \leq x_j)$, если таковая отсутствует, указывают [], cs - массив (вектор-столбец) длиной n , содержит верхнюю границу переменных $(cs_j \geq x_j)$, если таковая отсутствует, указывают [], k - целочисленная переменная используется, если в систему ограничений кроме неравенств входят и равенства, в матрице они будут находиться в k первых строках, оставшиеся l строк займут неравенства,

то есть $m=k+l$, x_0 - вектор-столбец начальных приближений длиной n .

Функция `linpro` возвращает массив неизвестных x , минимальное значение функции f и массив множителей Лагранжа kl [].

Рассмотрим использование функции `linpro` на примере решения следующей задачи линейного программирования.

ЗАДАЧА 13.2.

Найти такие значения переменных x_1, x_2, x_3, x_4 при которых функция цели L

$$L = -x_2 - 2x_3 + x_4$$

достигает своего минимального значения и удовлетворяются ограничения:

$$3x_1 - x_2 \leq 2$$

$$x_2 - 2x_3 \leq -1$$

$$4x_3 - x_4 \leq 3$$

$$5x_1 + x_4 \geq 6$$

$$x_1 \geq 0, x_2 \geq 0, x_3 \geq 0, x_4 \geq 0$$

Обратите внимание, что в четвертом ограничении присутствует знак \geq . Для приведения системы ограничений виду (13.6) необходимо четвертое уравнение умножить на -1. Решение задачи 13.1 представлено на листинге 13.4.

```
c=[0;-1;-2;1];
A=[3 -1 0 0;0 1 -2 0; 0 0 4 -1; -5 0 0 -1];
b=[2;-1;3;-6];
ci=[0;0;0;0];
[x,kl,f]=linpro(p,A,b,ai,[])
```

Листинг 13.4.

Полученные значения представлены на листинге 13.5.

f=

2.

kl=

0.

0.

0.

0.

0.0909091

1.0909091

0.5454545

0.4545455

x=

1.

1.

1.

1

Листинг 13.5.

Рассмотрим решение еще одной задачи линейного программирования.

ЗАДАЧА 13.3.

Туристическая фирма заключила контракт с двумя турбазами на одном из черноморских курортов, рассчитанных соответственно, на 200 и 150 человек.

Туристам для осмотра предлагается дельфинарий в городе, ботанический сад и походы в горы.

Составить маршрут движения туристов так, чтобы это обошлось возможно дешевле, если дельфинарий принимает в день 70 организованных туристов, ботанический сад 180, а в горы в один день могут пойти 110 человек.

Стоимость одного посещения выражается таблицей 13.2.

Таблица 13.2.

Турбаза	Дельфинарий	Ботанический сад	Поход в горы
1	5	6	20
2	10	12	5

Для решения задачи введем следующие обозначения:

x_1 число туристов первой турбазы, посещающих дельфинарий;

x_2 число туристов первой турбазы, посещающих ботанический сад;

x_3 число туристов первой турбазы, отправляющихся в поход;

x_4 число туристов второй турбазы, посещающих дельфинарий;

x_5 число туристов второй турбазы, посещающих ботанический сад;

x_6 число туристов второй турбазы, отправляющихся в поход.

Составим функцию цели, заключающуюся в минимизации стоимости мероприятий фирмы:

$$Z = 5x_1 + 6x_2 + 20x_3 + 10x_4 + 12x_5 + 5x_6.$$

Руководствуясь условием задачи, определим ограничения:

$$x_1 + x_4 \leq 70;$$

$$x_2 + x_5 \leq 180;$$

$$x_3 + x_6 \leq 110;$$

$$x_1 + x_2 + x_3 = 200;$$

$$x_4 + x_5 + x_6 = 150.$$

Кроме того, количество туристов, участвующих в мероприятиях не может быть отрицательным: $x_1 \geq 0, x_2 \geq 0, x_3 \geq 0, x_4 \geq 0, x_5 \geq 0, x_6 \geq 0$.

Решение задачи представлено в листинге 13.6. В массиве x будут храниться значения x_1, x_2, x_3, x_4, x_5 и x_6 . В матрице коэффициентов A , первые две строки отражают равенства системы ограничений, поэтому включен параметр $k=2$. Вектор ci представляет нижнюю границу неизвестных, то есть отражает тот факт, что они не могут быть меньше нуля. Если присвоить результат вычислений функции вектору из трех компонент, то первая из них будет содержать вектор неизвестных x , вторая — множители Лагранжа kl , а третья — минимальное значение функции цели f .

$$Z = [5; 6; 20; 10; 12; 5];$$

$$A = [1 \ 1 \ 1 \ 0 \ 0 \ 0;$$

$$0 \ 0 \ 0 \ 1 \ 1 \ 1; 1 \ 0 \ 0 \ 1 \ 0 \ 0; 0 \ 1 \ 0 \ 0 \ 1 \ 0; 0 \ 0 \ 1 \ 0 \ 0 \ 1];$$

$$b = [200; 150; 70; 180; 110];$$

$$ci = [0; 0; 0; 0; 0; 0];$$

$$k = 2;$$

$$[x, kl, f] = \text{linpro}(Z, A, b, ci, [], k);$$

Листинг 13.6.

Полученное в Scilab решение задачи представлено на листинге 13.7

```
f=
    2120.
kl=
    0.
    0.
   -20.
    0.
   -1.
    0.
   -6.
  -11.
    1.
    0.
    6.
x=
    30.
   170.
    0.
    40.
    0.
   110.
```

Листинг 13.7.

Авторы надеются, что разобрав примеры из этой главы, читатель сможет использовать Scilab для решения своих оптимизационных задач.

Этой главой мы завершаем первое знакомство с пакетом и надеемся, что свободно распространяемый пакет Scilab поможет читателю решать математические задачи, возникающие в его практической деятельности.