МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

(ДГТУ)

Кафедра «Математики и информатики»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
для проведения практических занятий
по дисциплине
«Прикладная математика»

Ростов-на-Дону
ДГТУ
2022

УДК 004.8(075.8)

Составитель: М.Н. Богачева

Методические указания для проведения практических занятий по дисциплине «Прикладная математика» – Ростов-на-Дону : Донской гос. техн. ун-т, 2022. – 76 с.

Методические указания предназначены для обучающихся по направлению подготовки магистратуры направления 090402 «Информационные системы и технологии».

УДК 004.8(075.8)

Печатается по решению редакционно-издательского совета
Донского государственного технического университета

Ответственный за выпуск зав. кафедрой «Математика и информатика» д-р физ.-мат. наук, профессор А.И. Сухинов

В печать _____ г.
Формат 60×84/16. Объем _____усл. п. л.
Тираж _____экз. Заказ № _____

Издательский центр ДГТУ
Адрес университета и полиграфического предприятия:
344000, г. Ростов-на-Дону, пл. Гагарина, 1

Тема 1. Численные методы решения уравнений

Уравнение называется алгебраическим, если его можно представить в виде:

$$\sum_{i=0}^{n} a_i x^i = 0 \qquad (1.1)$$

Формула (1.1) – каноническая форма записи алгебраического уравнения. Если уравнение $f(x)=0$ не удается привести к виду (1.1) заменой переменных, то уравнение называется трансцендентным.

Пример алгебраического уравнения: $2.5x^5 - 4x^4 - 3\cdot x^2 - 8x + 12 = 0.$

Вот пример трансцендентного уравнения.

$$x\ln x + x^2\cos x - \frac{x}{\lg x + 3} - 8 = 0.$$

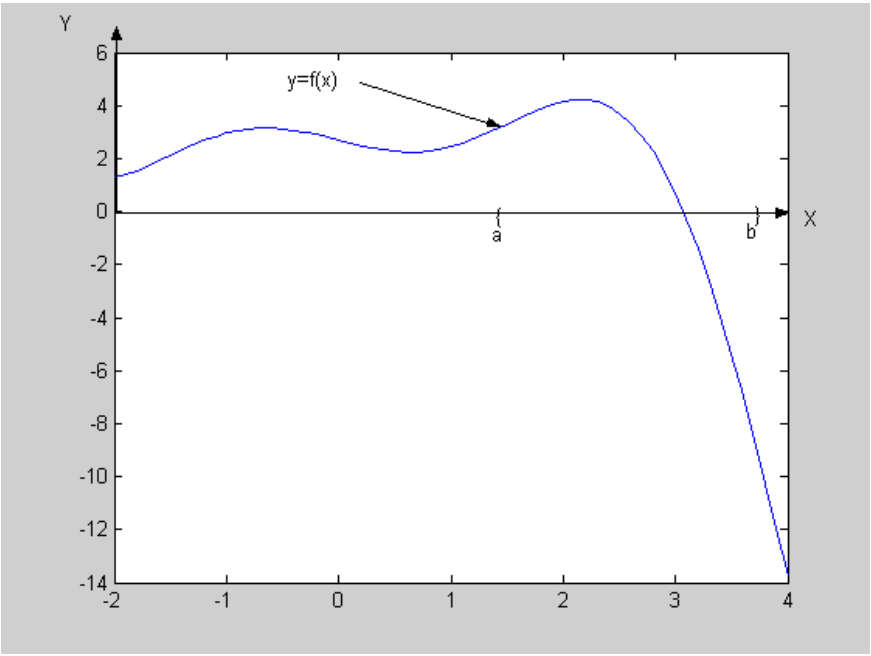
Решить уравнение означает найти такие значения *x*, при которых уравнение превращается в тождество.

Известно, что уравнение (1.1) имеет ровно *n* корней – вещественных или комплексных. Если *n*=1, 2, 3 [и иногда 4 (биквадратное уравнение)], то существуют точные методы решения уравнения (1.1). Если же *n*>4 или уравнение – трансцендентное, то таких методов, как правило, не существует, и решение уравнения ищут приближенными методами. Всюду при дальнейшем изложении будем предполагать, что *f*(*x*) – непрерывная функция. Методы, которые мы рассмотрим, пригодны для поиска некратных (то есть изолированных) корней.

1.1 Отделение корня

Решение уравнения состоит из двух этапов: 1 – отделение корня, 2 – его уточнение.

Отделить корень – значит указать такой отрезок [a, b] , на котором содержится ровно один корень уравнения $f(x)=0$.



Не существует алгоритмов отделения корня, пригодных для любых функций $f(x)$. Если удастся подобрать такие a и b, что

1) $f(a)\,f(b)<0$ (1.2)
2) $f(x)$ – непрерывная на [a, b] функция (1.3)
3) $f(x)$ – монотонная на [a, b] функция (1.4)

то можно утверждать, что на отрезке [a, b] корень отделен.

Условия (1.2) – (1.4) – достаточные условия того, что корень на [a, b] отделен, то есть если эти условия выполняются, то корень отделен, но невыполнение, например, условий (1.3) или (1.4) не всегда означает, что корень не отделен.

Корень можно отделить аналитически и графически.

Пример. Аналитически отделить положительный корень уравнения $x^3 - 7x - 5 = 0$

Решение. Составим таблицу

x	0	1	2	3
$y = x^3 - 7x - 5$	-5	-11	-11	1

1) $f(2)f(3)<0$,
2) $f(x)$ – непрерывная функция,
3) $y' = 3x^2 - 7 \geq 3 \cdot 4 - 7 > 0$, следовательно, $f(x)$ – монотонно возрастает на отрезке [2, 3].

Вывод: на отрезке [2, 3] *корень отделен*.

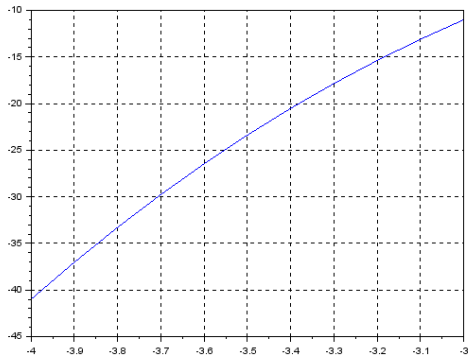
Графический метод отделения корня

С учетом того, что существует много программных продуктов для построения графиков функций, можно построить график в одном из них. Покажем, как это можно сделать в Scilab.

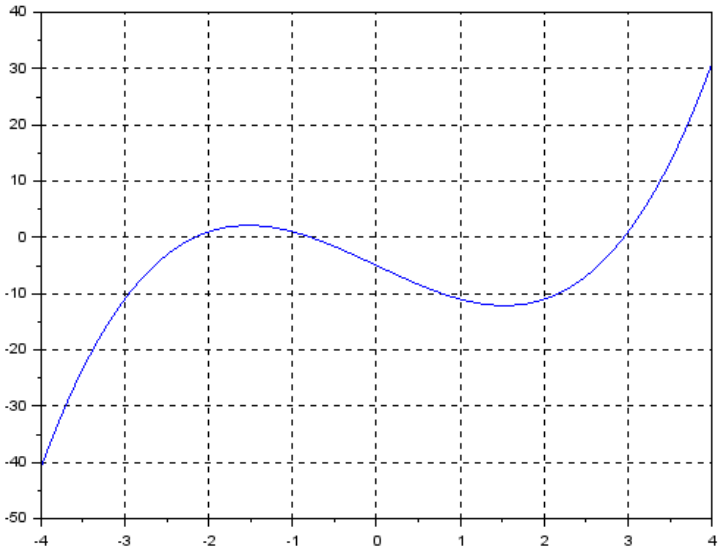
Пример: $x^3 - 7x - 5 = 0$

Составим программу:

```
clc
clf()
x=-4:.1:-3;y=x^3-7*x-5;
plot(x,y)
xgrid()
```



Из графика видно, что корня уравнения на отрезке [-4, -3] нет. Изменим пределы изменения аргумента, например, на следующие: [-4, 4]. Шаг можно оставить прежним. Получим:



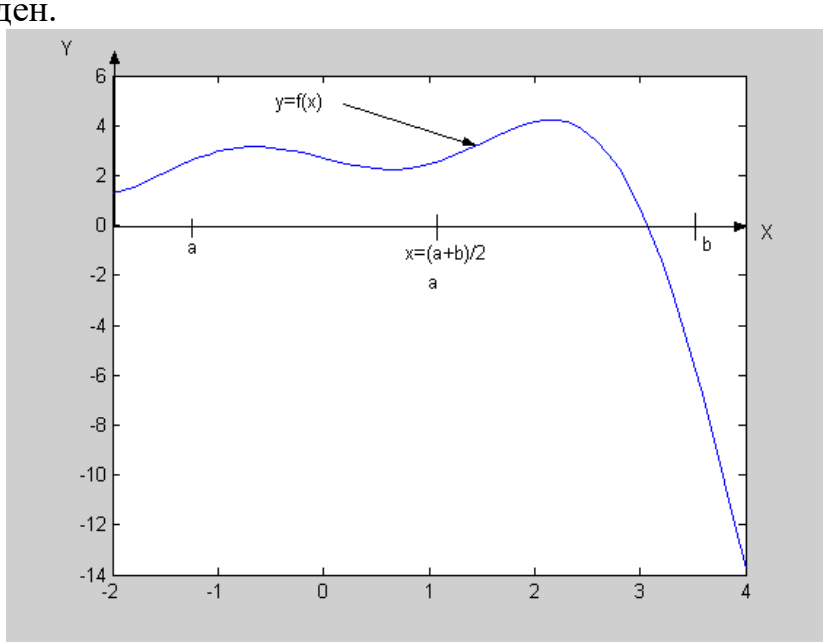
Из графика видно, что уравнение x^3-7x-5=0 имеет три корня и в качестве отрезков, на которых отделен корень уравнения, можно выбрать, например, такие: [-2.5, -1.5], [-1.5, 0.5], [2, 3].

1.2 Уточнение корня методом деления отрезка пополам

Уточнить корень – значит найти его приближенное значение с заданной погрешностью ε.

Самый простой метод, пригодный для любых непрерывных функций – метод деления отрезка пополам.

Предположим, что отрезок [a, b], на котором отделен корень уравнения, уже найден.



Пусть, например, *f*(a)>0, *f*(b)<0. Вычислим точку $x = \frac{a+b}{2}$. Если вместо корня взять точку *x*, то погрешность, которую мы при этом допустим, не превысит величины $\varepsilon_1 = \frac{b-a}{2}$. Если эта погрешность не превышает некоторую заданную погрешность ε, с которой нужно уточнить корень уравнения, то вычисления прекращаем и можно записать: $\xi = x \pm \frac{b-a}{2}$. В противном случае определяем новый отрезок [a, b], на котором отделен корень нашего уравнения. Для этого нам нужно знать знак функции в точке *x*. В нашем примере *f*(*x*)>0. Новый отрезок – отрезок [*x*, b], так как на концах этого отрезка функция имеет разные знаки. Переобозначим один из концов отрезка – в нашем случае положим a=x – и повторим процедуру для нового отрезка [a, b].

1.3 Метод касательных

Дополнительные предположения: *f*(x) дважды непрерывно дифференцируема на отрезке [a, b], на котором отделен корень; $f'(x)\,u\;f''(x)$ сохраняют постоянные знаки на отрезке [a, b]. Это – условия сходимости метода касательных. То есть выполнение этих условий гарантирует нам, что за определенное число шагов (вычислений по формуле (1.6)) мы уточним корень с любой заданной погрешностью ε. Вот алгоритм метода касательных.

За x_0 выбирается точка, в которой выполняется условие:

$$f(x_0)\,f''(x_0) > 0 \qquad (1.5)$$

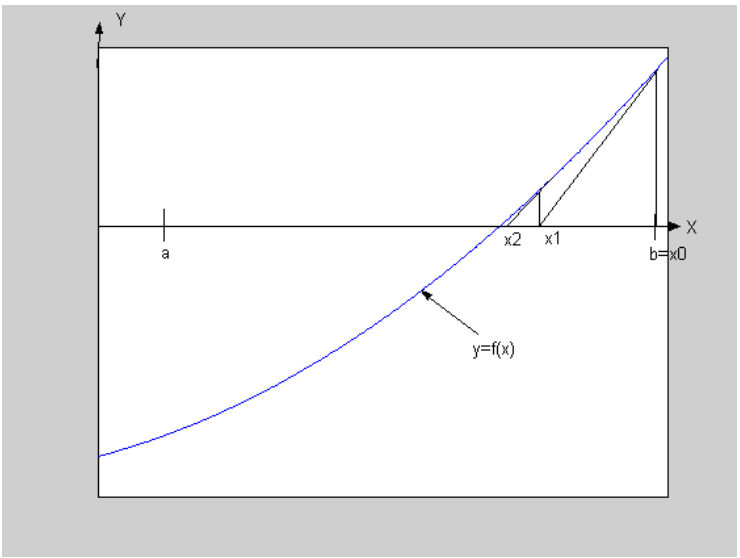
Это либо точка a, либо точка b. Далее вычисляются точки

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \qquad (1.6)$$

до тех пор, пока не выполнится условие

$$\left|x_{n+1} - x_n\right| < \varepsilon \qquad (1.7)$$

Тогда x_{n+1} - приближенное значение корня с погрешностью ε.



Проверим условия сходимости для нашего уравнения.

$$x^3 - 7x - 5 = 0. f(x) = x^3 - 7x - 5.$$

$$f'(x) = 3x^2 - 7.$$

$$f''(x) = 6x.$$

Предположим, мы хотим уточнить корень уравнения, отделенный на отрезке [2, 3]. Минимальное значение первой производной достигается в точке x=2 и оно положительно. Вторая производная положительна при любом x>0. В видеолекции https://youtu.be/U2787R9nlUY рассмотрен другой способ проверки условий сходимости. Там же показано, как можно выполнить 1 задание (в нашем курсе задания на эту тему – задания 3 и 4).

Далее. Выбираем точку x_0. Это либо точка a (2 в нашем случае), либо точка b (3). Выясним, какая именно. Проверим, в какой из точек выполняется условие:

$$f(x_0) f''(x_0) > 0$$

В точке a=2 $f(a)<0$, в точке b=3 $f(b)>0$:

$$f(2) = 2^3 - 7 \cdot 2 - 5 = 8 - 14 - 5 = -11 < 0. f(3) = 3^3 - 7 \cdot 3 - 5 = 27 - 21 - 5 = 1 > 0.$$

Поэтому $x_0=3$. $f(3) f''(3) > 0$.

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 3 - \frac{f(3)}{f'(3)} = 3 - \frac{3^3 - 7 \cdot 3 - 5}{3 \cdot 3^2 - 7} = 3 - \frac{1}{20} = 2.95.$$

Предположим, что погрешность, с которой мы хотим уточнить корень, равна ε=0.1.

$|x_1 - x_0| = |2.95 - 3| = 0.05 < \varepsilon = 0.1$, поэтому x_1 можно считать корнем уравнения с заданной погрешностью: $\xi = 2.95 \pm 0.1$. Если бы это оказалось не так, то нужно было найти x_2, вычислить разность $|x_2-x_1|$ и если она меньше ε, то x_2 – корень уравнения с заданной погрешностью и т.д.

Для решения уравнений, в том числе трансцендентных, в Scilab применяют функцию fsolve(x0,f), где x0 - начальное приближение, f - функция, описывающая левую часть уравнения f(x)=0.

Покажем, как можно найти все три корня нашего уравнения.

```
clc
function y=f(x), y=x^3-7*x-5;  endfunction
[a,b]=fsolve([-2,-1,3],f);disp([a;b])
-2.1660127  -0.7828157   2.9488284
-1.776D-14   1.510D-14   3.553D-14
```

Здесь в качестве начальных приближений выбраны точки -2, -1 и 3.

Если нам нужен только один корень уравнения, например, ближайший к точке 3, то изменяем оператор fsolve: [a,b]=fsolve(3,f). Запустив измененную программу на выполнение, получим:

```
2.9488284
3.553D-15
```

Первое число – это корень уравнения, второе – значение левой части уравнения в этой точке ($3.553 \cdot 10^{-15}$).

Решить алгебраическое уравнение в Scilab можно и с помощью функции roots.

В нашем примере: x^3-7x-5=0. Задаем вектор коэффициентов полинома, идущих в порядке уменьшения степеней: сначала коэффициент перед x^3, потом перед x^2 и т.д. Затем используем функцию roots:

```
--> a=[1 0 -7 -5];x=roots(a)
 x  =
   2.9488284
  -2.1660127
  -0.7828157
```

Систему нелинейных уравнений также можно решить, используя функцию **fsolve.**

$$\begin{cases} x^2 + y^2 = 1 \\ x^3 - y - 2xy = 0 \end{cases}$$

```
clc
function [y]=ff(x)
 y(1)=x(1)^2+x(2)^2-1;
 y(2)=x(1)^3-x(2)-2*x(1)*x(2);
endfunction
[t,value]=fsolve([-.5;-.5],ff);
disp([t,value])

  -0.4492543   0.
  -0.8934039  -8.771D-15
```

Однако, как и при решении уравнения, точность решения часто зависит от того, насколько удачно мы задали начальное приближение x_0. В разделе 1.4 показано, как можно задать начальное приближение при решении систем двух нелинейных уравнений.

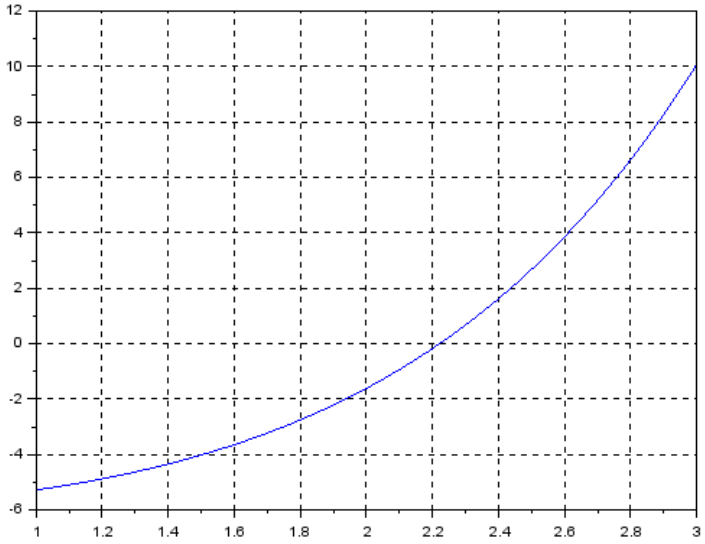
Вот пример решения уравнения рассмотренными методами.

Пример. Решить уравнение $e^x-x-7=0$

Решение.

1 этап. Отделяем корень. Для этого строим график функции y= e^x-x-7

```
clc
clf()
function y=f(x)
  y=exp(x)-x-7
endfunction
x=1:.01:3;
plot(x,f(x));xgrid()
```



Из графика видно, что корень отделен на отрезке [2, 3].

2 этап. Уточнение корня с погрешностью ε=0.05.

Метод деления отрезка пополам

Обозначим $f(x)=e^x-x-7$.

$f(2)=e^2-2-7=-1.6109439$; $f(3)=e^3-3-7=10.085537$

$x=(2+3)/2=2.5$ Если взять точку x в качестве корня, то погрешность не превысит (3-2)/2=0.5.

$f(2.5)=e^{2.5}-2.5-7=2.682494$. Корень находится на отрезке [2, 2.5] так как на концах отрезка функция имеет разные знаки.

$x=(2+2.5)/2=2.25$ $\xi=2.25\pm0.25$ $f(2.25)=e^{2.25}-2.25-7=0.2377358$ Корень находится на отрезке [2, 2.25] так как на концах отрезка функция имеет разные знаки. $x=(2+2.25)/2=2.125$ $\xi=2.125\pm0.125$ и т.д., пока погрешность не станет меньше или равна заданной.

Метод касательных

Проверим условия сходимости.

$$f'(x)=e^x-1>0 \quad \forall\, x\in[2,3]$$

$$f''(x)=e^x>0$$

$x_0=3$ т.к. $f(3)\,f''(3)>0$ $x_1=x_0-f(x_0)/f'(x_0)=3-(e^3-3-7)/(e^3-1)=2.47$.

$|x_1-x_0|=0.53>eps=0.1$ $x_2=x_1-f(x_1)/f'(x_1)=2.47-(e^{2.47}-2.47-7)/(e^{2.47}-1)=2.25$

$|x_2-x_1|=0.22>eps=0.05$ $x_3=x_2-f(x_2)/f'(x_2)=2.25-(e^{2.25}-2.25-7)/(e^{2.25}-1)=2.22$

$|x_3-x_2|=0.03<eps=0.05$

Ответ: корень уравнения $\xi=2.22\pm0.05$

1.4. Решение системы двух нелинейных уравнений в среде Scilab. Функция plotimplicit

Для решения систем уравнений (как и для решения уравнений) в Scilab используют функцию *fsolve*. Ее синтаксис может быть таким:

[a,b]=fsolve(x0,f).

Здесь x0 – начальное приближение, f – уравнение или специальным образом записанная система уравнений, которое (ую) необходимо решить, **a** – решение уравнения или системы уравнений: (скалЯр), если решается одно уравнений или вектор, если решается система уравнений. От того, насколько удачно мы зададим начальное приближение, часто зависит то, что мы получим. Если x_0 задан неудачно, а иногда мы задаем его наугад, то, проделав определенное число итераций по встроенному алгоритму решения уравнения/системы уравнений, компьютер выдаст ответ – число или вектор **a**, который не является корнем уравнения. Судить о том, получили мы корень или нет, можно по значению **b**. Об этом мы поговорим чуть позже. Кроме того, в результате мы даже в лучшем случае получим только один ответ, хотя система уравнений (будем теперь говорить только о ней) может иметь несколько решений. Покажем, как можно решить систему двух уравнений на примере:

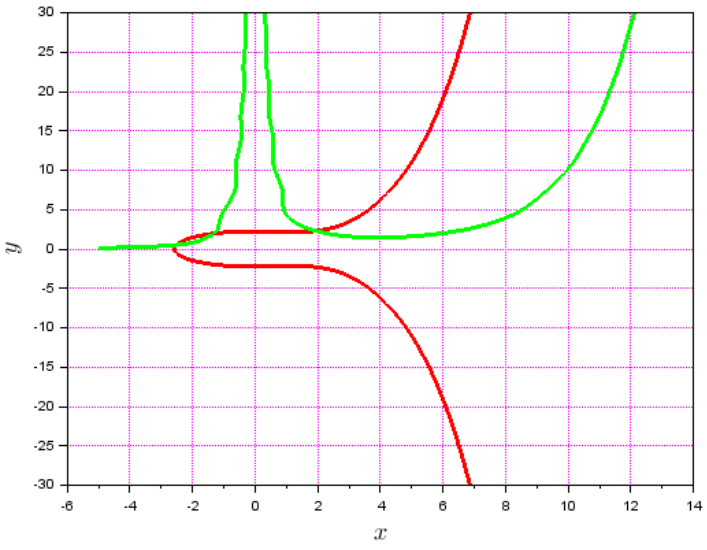
Пример. Решить систему нелинейных уравнений:

$$\begin{cases} x^2 + y^2 + x = e^x + 4 \\ \sqrt[3]{x+y} + x^2 y = x\sin y + 5 + 2^x \end{cases}$$

В версии Scilab 6.1.0 появилась возможность строить графики функций, заданных неявно, то есть в виде: $f(x,y)=0$. Это можно сделать с помощью функции plotimplicit. Составим программу:

```
clf();clc;clear
equation = "x^2+y^2+x=exp(x)+4"
plotimplicit(equation, -5:0.1:20, -30:0.1:30, "r")
gce().children.thickness =3
equation_1 = "(x+y)^(1/3)+x^2*y=x*sin(y)+5+2^x"
plotimplicit (equation_1, -5:0.1:20, -30:0.1:30, "g")
xgrid(color("magenta"),1,7);gce().children.thickness =3;title("")
```

Значения переменных x и y при построении графиков выбраны произвольно. При необходимости мы бы их скорректировали.



Каждая точка, например, красной кривой – это решение первого уравнения системы. Каждая точка зеленой – решение второго уравнения. Точки пересечения кривых – решения системы уравнений. Теперь мы можем обоснованно задать начальное приближение x_0 для нахождения любого из трех (в данном случае) решений.

Далее. Задаем систему уравнений, которую мы хотим решить. Она должна быть переписана в виде:

$Y_1(x(1),x(2))=0$

$Y_2(x(1),x(2))=0$

В нашем случае это можно сделать так. Пусть имя нашей функции ff. Аргументом является вектор x, состоящий из двух компонент x(1) (в условии примера это буква x) и x(2) (в условии это буква y). Вектор Y состоит из двух строк. Первая из них – левая часть первого уравнения, переписанного в виде $x^2 + y^2 + x - e^x - 4 = 0$, где вместо x стоит x(1), а вместо y - x(2), вторая – левая часть второго уравнения $\sqrt[3]{x+y} + x^2 y - x\sin y - 5 - 2^x = 0$ с такой же заменой переменных:

```
function [y]=ff(x)
  y(1)= x(1)^2+x(2)^2+x(1)-exp(x(1))-4;
  y(2)= (x(1)+x(2))^(1/3)+x(1)^2*x(2)-x(1)*sin(x(2))-5-2^x(1);
endfunction
```

Для поиска первого решения зададим такое начальное приближение: [-2;0], то есть начальное значение x=-2, начальное значение y=0.

```
[t,d]=fsolve([-2;0],ff)
printf("Решение системы уравнений x= %2.5f  y= %2.5f\n",t(1),t(2))
printf(" Левая часть первого уравнения равна  %2.12f\n  ",d(1))
printf(" Левая часть второго уравнения равна  %2.12f  ",d(2))
```

t(1) – это x, t(2) – это y (это решение системы уравнений)

d(1) – это значение левой части уравнения $x^2 + y^2 + x - e^x - 4 = 0$, d(2) – значение левой части уравнения $\sqrt[3]{x+y} + x^2 y - x\sin y - 5 - 2^x = 0$. Соединив все части программы

```
clf()
clc
clear
equation = "x^2+y^2+x=exp(x)+4"
plotimplicit(equation, -5:0.1:20, -30:0.1:30, "r")
gce().children.thickness =3
equation_1 = "(x+y)^(1/3)+x^2*y=x*sin(y)+5+2^x"
plotimplicit (equation_1, -5:0.1:20, -30:0.1:30, "g")
xgrid(color("magenta"),1,7)
gce().children.thickness =3
title("")
function [y]=ff(x)
```

```
  y(1)= x(1)^2+x(2)^2+x(1)-exp(x(1))-4;
  y(2)= (x(1)+x(2))^(1/3)+x(1)^2*x(2)-x(1)*sin(x(2))-5-2^x(1);
endfunction
[t,d]=fsolve([-2;0],ff)
printf("Решение системы уравнений x= %2.5f  y= %2.5f\n",t(1),t(2))
printf(" Левая часть первого уравнения равна  %2.12f\n  ",d(1))
printf(" Левая часть второго уравнения равна  %2.12f  ",d(2))
```

и запустив ее на выполнение, получаем:

```
Решение системы уравнений x= -2.51484  y= 0.52087
Левая часть первого уравнения равна  0.000000000000
Левая часть второго уравнения равна  -0.000000000000
```

Близость к нулю значений d(1) и d(2) свидетельствует о том, что мы получили именно решение нашей системы уравнений. Поменяв теперь значения начального приближения и запустив программу на выполнение, найдем другие решения нашей системы уравнений.

```
[t,d]=fsolve([-1;2],ff)

Решение системы уравнений x= -1.30065  y= 1.97010
 Левая часть первого уравнения равна  -0.000000000000
  Левая часть второго уравнения равна  -0.000000000000

[t,d]=fsolve([2;2],ff)

Решение системы уравнений x= 1.94603  y= 2.29516
 Левая часть первого уравнения равна  0.000000000000
  Левая часть второго уравнения равна  0.000000000000
```

Тема 2. Аппроксимация

Аппроксимация – замена одной функции $f(x)$ другой, похожей функцией $Q(x)$. Например, функцию, полученную экспериментально в виде таблицы или графика, надо записать в аналитическом виде, либо функцию, достаточно сложную нужно заменить похожей, но более простой.

Простейший способ аппроксимации – замена функции $f(x)$ алгебраическим полиномом

$$f(x) \approx c_0 + c_1 x + c_2 x^2 + \ldots + c_n x^n = Q(x, c_j) \qquad (2.1)$$

Аппроксимацию называют точечной, если $f(x)$ задана на конечном множестве точек (узлов) $x_0, x_1, \ldots, x_m$.

Если $f(x)$ задана на непрерывном множестве значений аргументов, то аппроксимацию называют интегральной.

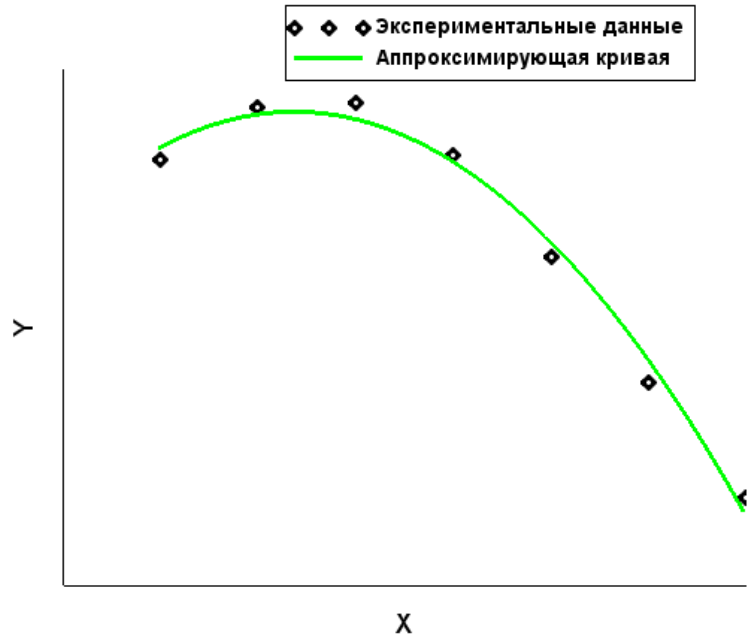


Рис. 2.1. Точечная аппроксимация

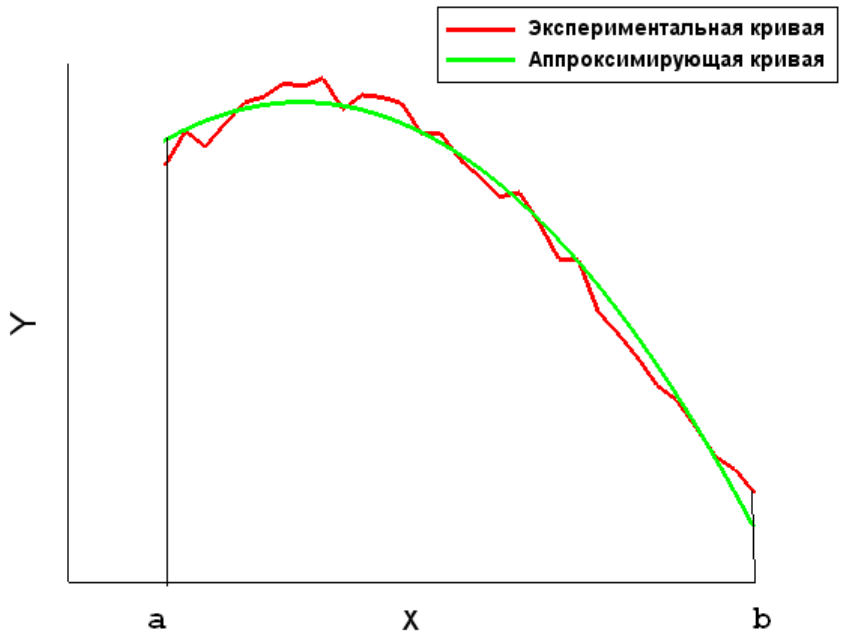


Рис. 2.2. Интегральная аппроксимация

Необходимо так подобрать коэффициенты аппроксимирующей кривой, чтобы $Q(x)$ как можно меньше отличалась от $f(x)$.

Мерой погрешности аппроксимации назовем некоторое неотрицательное число $\mu(f, Q)$. Естественно потребовать, чтобы $\mu(f,Q)=0$, если $f(x)$ и $Q(x,c_j)$ совпадают на общем для их задания множестве значений аргументов. Справедливо и обратное утверждение.

После того, как мера каким-либо образом задана, необходимо указать способ получения коэффициентов c_j. Наиболее часто используемыми подходами к решению задачи аппроксимации являются метод наименьших квадратов (МНК), метод наименьших модулей и минимаксный подход.

2.1.1. Метод наименьших квадратов

2.1.1.1. Аппроксимация полиномами

Чаще всего при точечной аппроксимации используют меру

$$K = \sum_{i=0}^{m} [f(x_i) - Q(x_i, c_j)]^2 \tag{2.2}$$

а коэффициенты c_j ищут из условия

$$K \to \min_{c_j} \tag{2.3}$$

Это точечная квадратичная аппроксимация.

При интегральной аппроксимации

$$K = \int_a^b [f(x) - Q(x, c_j)]^2 \, dx$$

$$K \to \min_{c_j}$$

Описанный подход к задаче аппроксимации называется методом наименьших квадратов. Условия (2.2) и (2.3) геометрически означают: из всех кривых заданного вида выбирают ту, у которой сумма площадей квадратов отклонений – наименьшая. В математике кривую $y=Q(x,c_j)$ называют регрессией Y на X. Это означает, что у функции $y=Q(x,c_j)$ число искомых коэффициентов меньше количества узлов, в крайнем случае возможно равенство $n+1 \le m+1$. Здесь n степень аппроксимирующего полинома, а m – номер последнего узла (нумерация ведется с нуля).

Если аргументом считать y, а x – функцией, то говорят о регрессии X на Y. Отклонения в этом случае откладывают по оси X.

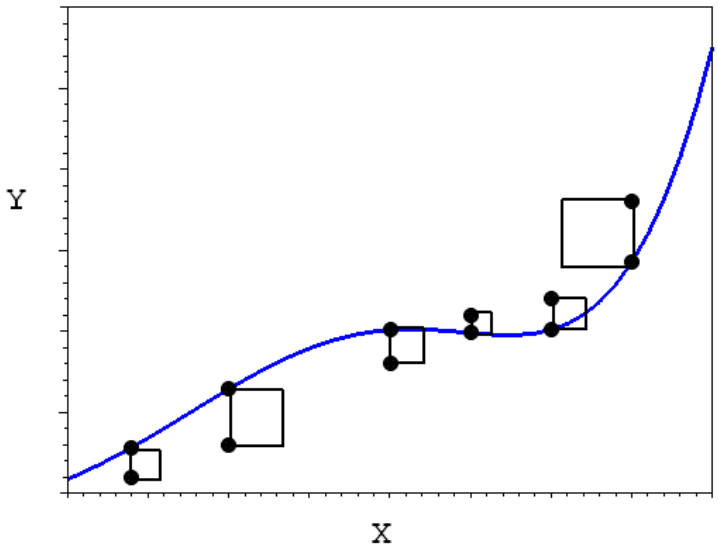


Рис. 2.3. Регрессия Y на X

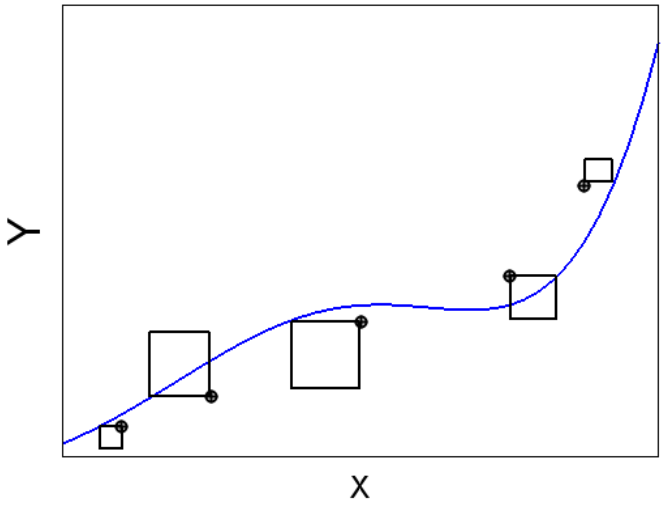


Рис. 2.4. Регрессия X на Y

Покажем, как находятся коэффициенты c_j при точечной аппроксимации методом наименьших квадратов.

Минимальное значение K в формуле (2.2) достигается при тех c_j, при которых

$$\frac{\partial K}{\partial c_j} = 0, \quad j = 0,1,...,n \tag{2.4}$$

Имеем:

$$-2\sum_{i=0}^{m}[f(x_i) - Q(x_i, c_j)]\frac{\partial Q(x_i, c_j)}{\partial c_j} = 0, \quad j = 0,1,...,n \tag{2.5}$$

Здесь $Q(x, c_j) = c_0 + c_1 x + c_2 x^2 + ... + c_n x^n$.

Обозначим $y_i = f(x_i)$. Тогда система уравнений (2.4) примет вид (2.6):

$$\begin{cases} \sum_{i=0}^{m}(y_i - c_0 - c_1 x_i - c_2 x_i^2 - ... - c_n x_i^n) = 0 \\ \sum_{i-0}^{m}(y_i - c_0 - c_1 x_i - c_2 x_i^2 - ... - c_n x_i^n)x_i = 0 \\ ..... \\ \sum_{i=0}^{m}(y_i - c_0 - c_1 x_i - c_2 x_i^2 - ... - c_n x_i^n)x_i^n = 0 \end{cases} \tag{2.6}$$

Окончательно

$$\begin{cases} \sum_{i=0}^{m} y_i = (m+1)c_0 + c_1\sum_{i=0}^{m} x_i + ... + c_n\sum_{i=0}^{m} x_i^n \\ \sum_{i=0}^{m} x_i y_i = c_0\sum_{i=0}^{m} x_i + c_1\sum_{i=0}^{m} x_i^2 + ... + c_n\sum_{i=0}^{m} x_i^{n+1} \\ ..... \\ \sum_{i=0}^{m} x_i^n y_i = c_0\sum_{i=0}^{m} x_i^n + c_1\sum_{i=0}^{m} x_i^{n+1} + ... + \sum_{i=0}^{m} x_i^{2n} \end{cases} \tag{2.7}$$

Здесь x_i и y_i – заданные числа.

Пример 2.1: Методом наименьших квадратов построить аппроксимирующую параболу для функции

x	0	1	2	3
y	-1,2	-0,1	1,3	1,8

Составим таблицу

x	y	xy	x^2y	x^2	x^3	x^4	Q(x,cj)
0	-1,2	0	0	0	0	0	-1,26
1	-0,1	-0,1	-0,1	1	1	1	0,08
2	1,3	2,6	5,2	4	8	16	1,12
3	1,8	5,4	16,2	9	27	81	1,86
6	**1,8**	**7,9**	**21,3**	**14**	**36**	**98**	

Запишем систему нормальных уравнений для нашей кривой.

$$\begin{cases} \sum_{i=0}^{3} y_i = 4c_0 + c_1\sum_{i=0}^{3} x_i + c_2\sum_{i=0}^{3} x_i^2 \\ \sum_{i=0}^{3} x_i y_i = c_0\sum_{i=0}^{3} x_i + c_1\sum_{i=0}^{3} x_i^2 + c_2\sum_{i=0}^{3} x_i^3 \\ \sum_{i=0}^{3} x_i^2 y_i = c_0\sum_{i=0}^{3} x_i^2 + c_1\sum_{i=0}^{3} x_i^3 + c_2\sum_{i=0}^{3} x_i^4 \end{cases} \text{ или } \begin{cases} 1{,}8 = 4c_0 + 6c_1 + 14c_2 \\ 7{,}9 = 6c_0 + 14c_1 + 36c_2 \\ 21{,}3 = 14c_0 + 36c_1 + 98c_2 \end{cases}$$

Решая систему, получаем:

$c_0 = -1{,}26, \quad c_1 = 1{,}48, \quad c_2 = -0{,}15$ или $y = -1{,}26 + 1{,}49x - 0{,}15x^2$

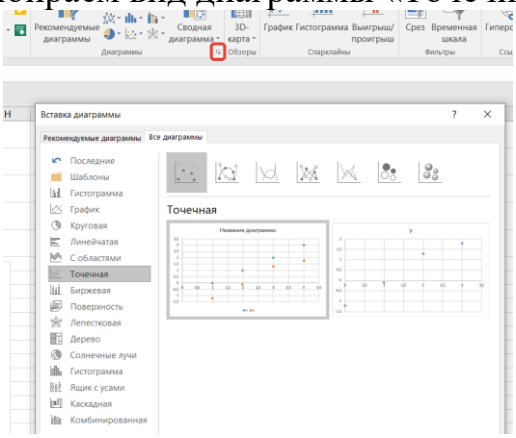
Вычисляя K по формуле (2.2), получаем: K=0,0720. У любой параболы второй степени значение K, вычисленное по формуле (2.2), будет больше, чем 0,0720.

Решение задачи аппроксимации в Excel

Одним из самых простых и быстрых способов решения задачи аппроксимации является решение задачи в Excel.

Excel предлагает такие виды аппроксимирующей кривой (MS Office 2016): экспоненциальная ($y=ae^{bx}$), линейная ($y=a+bx$ – установлена по умолчанию), логарифмическая ($y=a+b\ln x$), полиномиальная (степень полинома задает пользователь), степенная ($y=ax^b$). Предусмотрена также линейная фильтрация.

Покажем, как можно решить пример 2.1 в Excel. Вводим исходные данные. Выделяем столбцы x и y. На вкладке «Вставка» в раскрывающемся меню «Диаграммы» выбираем вид диаграммы «Точечная».



Получаем график, на который нанесены исходные данные (рис. 2.5).

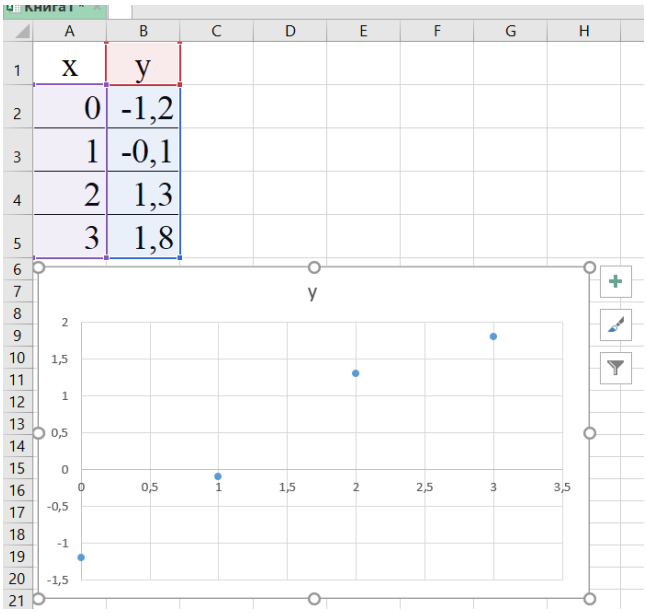


Рис. 2.5. Исходные данные на графике в Excel

Щелкнув правой кнопкой мыши по одной из синих точек, в раскрывающемся подменю выбираем пункт «Добавить линию тренда». В появившемся диалоговом окне «Формат линии тренда» (рис. 2.6) выбираем нужный тип кривой.

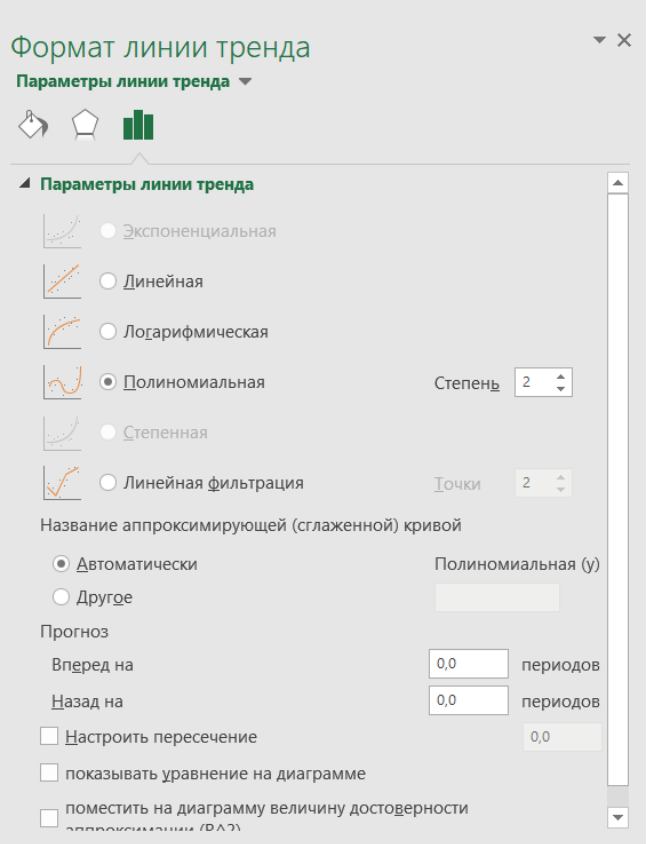


Рис. 2.6. Формат линии тренда

В нашем случае это полиномиальная кривая. Поставив точку слева от нужной надписи, выбираем степень аппроксимирующего полинома. По

умолчанию стоит степень 2, которую будем использовать для расчетов. Поставив галочки слева от надписей «Показывать уравнение на диаграмме» и «Поместить на диаграмму величину достоверности аппроксимации» (о коэффициенте R^2 см. ниже), после редактирования диаграммы получим линию тренда (рис.2.7).

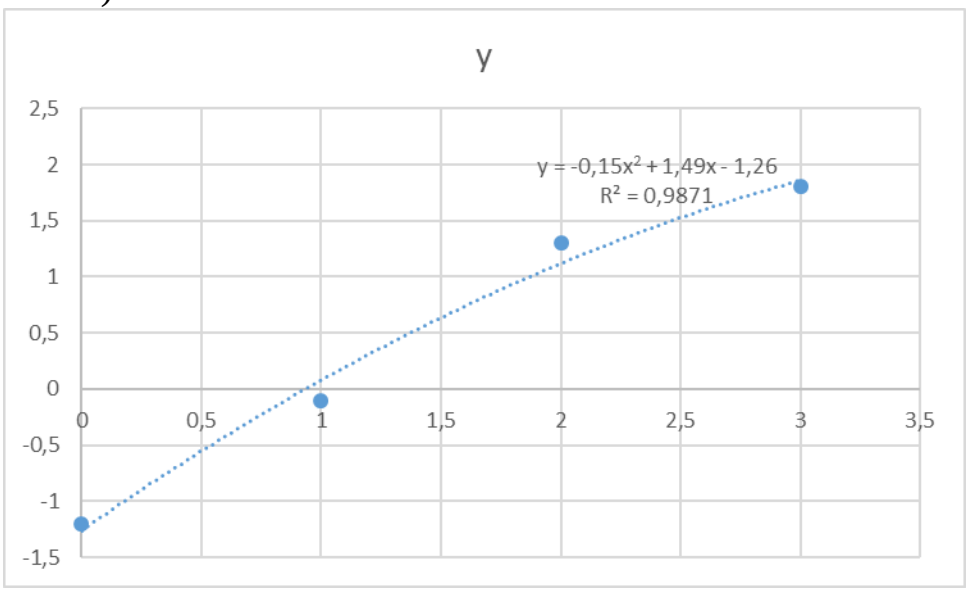


Рис. 2.7. Исходные данные и линия тренда

Убеждаемся, что получившееся уравнение совпадает с приведенным на с. 14.

Уравнения других типов кривых получаются аналогично.

При решении задачи аппроксимации в Excel выдается коэффициент достоверности аппроксимации (детерминации)

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(Y_i - y_i)^2}{\sum_{i=1}^{n}(Y_i - \overline{Y})^2} = 1 - \frac{\sum_{i=1}^{n}(Y_i - y_i)^2}{\sum_{i=1}^{n}Y_i^2 - \frac{1}{n}\left(\sum_{i=1}^{n}Y_i\right)^2}$$

Здесь Y_i – исходные данные, y_i – значение аппроксимирующей кривой в точках таблицы.

Решение задачи аппроксимации в среде Scilab

Для поиска коэффициентов $\boldsymbol{c}_i$ в формуле (2.1) в Scilab используется функция *datafit*:

[a, S]=datafit(F,z,c)

Здесь *F* –специальным образом заданная функция.

Для решения задачи задается вектор начальных приближений $\boldsymbol{c}$. Размерность вектора равна количеству искомых коэффициентов. *z* – матрица исходных данных размерности 2 x *n*, состоящая из *n* пар значений (x_i, y_i).

Значение функции *F* в точке x_i равно невязке:

$$F(c, x_i) = y_i - c_0 - c_1 x_i - c_2 x_i^2 - \ldots - c_n x_i^n .$$

Результатом работы функции *datafit* являются искомые коэффициенты (они обозначены как вектор ***a***) и *S* – сумма площадей квадратов отклонений (см. формулу 2.2). Начиная с версии Scilab 6.1.0 смысловая нагрузка выходного параметра S изменилась. В приведенных ниже программах значения суммы квадратов отклонений вычислялись непосредственно по формуле 2.2.

Вот решение примера 2.1 с использованием функции *datafit*:

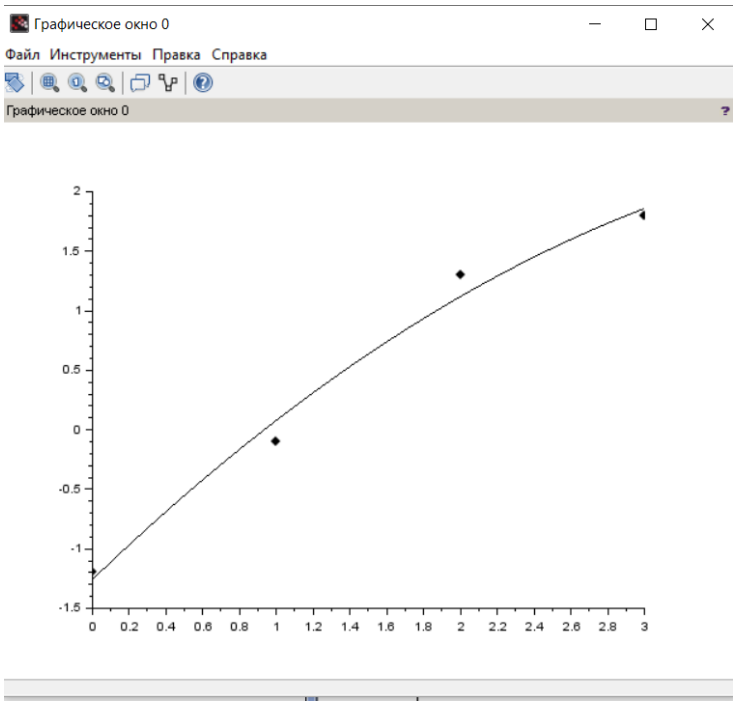
Программа на Scilab:

```
clf();clc
//Функция, вычисляющая разность между экспериментальными
//и теоретическими значениями,
//перед использованием необходимо определить
//z=[x;y] - матрицу исходных данных и
//c - вектор начальных значений коэффициентов,
//размерность вектора должна совпадать
//с количеством искомых коэффициентов
function y=G(c,z)
y=z(2)-c(1)-c(2)*z(1)-c(3)*z(1)^2
endfunction
//Исходные данные
x=[0 1 2 3];y=[-1.2 -0.1 1.3 1.8];
//Формирование матрицы исходных данных
z=[x;y];//та же буква, что и в функции y=G(c,z)
//Вектор начальных приближений
c=[0;0;0];//нулей столько, сколько искомых коэффициентов. Это нач. приближение
a=datafit(G,z,c); //Решение задачи
//Нанесение на график экспериментальных данных
plot2d(x,y,-4);
//Построение графика функции на отрезке [xmin , xmax]
t=min(x):0.01:max(x);Ptc=a(1)+a(2)*t+a(3)*t^2;plot2d(t,Ptc);
err=sum((y-a(1)-a(2)*x-a(3)*x.^2)^2)//Сумма квадратов отклонений
disp("err=", err)
disp("a=", a)
```

Запустив программу на выполнение, получаем:

```
"err="
  0.0720000
 "a="
 -1.2600001
  1.4900003
 -0.1500001
```

Ниже приведен график получившейся кривой.



2.1.1.2. Неполиномиальная регрессия

Часто на практике возникает следующая задача. Имеется объект исследования (ОИ), который характеризуется набором переменных: входных $(x_i \quad i=1,2,...,k)$ и выходной y.

Рис 2.8. Схема объекта исследования

Требуется найти зависимость выходной переменной от входных

$$y = f(x_1, x_2, ..., x_k). \tag{2.8}$$

При этом считается, что механизмы процессов, протекающих внутри объекта исследования, неизвестны, а имеются только соответствующие значения входных и выходных параметров. Такая задача носит название задачи «черного ящика».

Рассмотрим простейший случай, когда на вход действует только одна переменная x и требуется найти

$$y = f(x). \tag{2.9}$$

Решение задачи моделирования (то есть построение зависимости (2.9)) в этом случае состоит из четырех этапов:

- проведение эксперимента;
- выбор вида экспериментальной зависимости;
- нахождение параметров выбранной зависимости;
- исследование модели и выводы.

На первом этапе задаем значения входной переменной *x* из возможного диапазона и замеряем соответствующие значения выходной переменной *y*. Получаем таблицу:

x	x_0	x_1	…	x_m
y	y_0	y_1	…	y_m

По экспериментальным данным строим график (рис. 2.9).

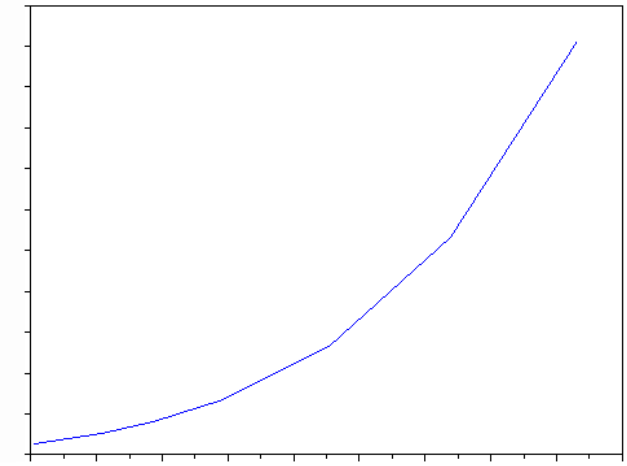


Рис. 2.9. Экспериментальная кривая

На втором этапе необходимо выбрать формулу, которая могла бы описать экспериментальные данные. Во втором столбце таблицы 2.1 приведены основные типовые формулы, наиболее часто встречающиеся в задачах химии и химической технологии.

Таблица 2.1

Определение формы регрессионной кривой

№	Формула	Средние точки	
1	$y = ax^b$	$\sqrt{x_0 x_m}$	$\sqrt{y_0 y_m}$
2	$y = ab^x$	$\frac{x_0 + x_m}{2}$	$\sqrt{y_0 y_m}$
3	$y = \frac{1}{a + bx}$	$\frac{x_0 + x_m}{2}$	$\frac{2y_0 y_m}{y_0 + y_m}$
4	$y = a + b\lg x$	$\sqrt{x_0 x_m}$	$\frac{y_0 + y_m}{2}$
5	$y = a + \frac{b}{x}$	$\frac{2x_0 x_m}{x_0 + x_m}$	$\frac{y_0 + y_m}{2}$
6	$y = \frac{ax}{b + x}$	$\frac{2x_0 x_m}{x_0 + x_m}$	$\frac{2y_0 y_m}{y_0 + y_m}$

Здесь (x_0, y_0) – первая, а (x_m, y_m) – последняя точки исходной таблицы. Для каждой функции рассчитывается средняя точка. Все эти точки наносятся на график с экспериментальной кривой. Выбирается та кривая, средняя точка которой находится ближе всего к экспериментальной кривой. Этот метод выбора вида зависимости называется *методом средних точек*. Покажем, как решается эта часть задачи на конкретном примере.

Пример 2.2. Зависимость давления насыщенного пара бензола от температуры выражается следующими экспериментальными данными:

T, K	*270,5*	*280,8*	*288,6*	*299,2*	*315,4*	*333,8*	*353,2*
$p \cdot 10^{-5}$, *Па*	*0,0267*	*0,0533*	*0,0800*	*0,1333*	*0,2667*	*0,5333*	*1,0133*

Методом средних точек определить вид зависимости. Параметры выбранной зависимости подобрать методом наименьших квадратов.

Решение. Для уменьшения вычислительной погрешности сделаем преобразование $x = \dfrac{T - 200}{100}$.

$x = \dfrac{T-200}{100}$	0,705	0,808	0,886	0,992	1,154	1,338	1,532
$p \cdot 10^{-5}$, *Па*	0,0267	0,0533	0,0800	0,1333	0,2667	0,5333	1,0133

Тогда

xap=(0,705 +1,532)/2= 1,1185

xgeo=$(0{,}705*1{,}532)^{0.5}$ = 1,0392594

xgar=2*0,705*1,532/(0,705+1,532)= 0,9656325

yap=(0,0267+1,0133)/2= 0,52

ygeo =$(0{,}0267*1{,}0133)^{0.5}$= 0,1644844

ygar =2*0,0267*1,0133/(0,0267+1,0133)= 0,0520291

Средняя точка для первой кривой –(1,0392594; 0,1644844)

Средняя точка для второй кривой –(1,1185; 0,1644844)

Средняя точка для третьей кривой –(1,1185; 0,0520291)

Средняя точка для четвертой кривой –(1,0392594; 0,52)

Средняя точка для пятой кривой –(0,9656325; 0,52)

Средняя точка для шестой кривой –(0,9656325; 0,0520291)

Нанесем на график исходные данные (x и y) и соединим соседние точки отрезками прямых. На этот же график нанесем средние точки для каждой кривой, снабдив их соответствующими номерами (рис. 2.10).

Из графика видно, что ближе всего к экспериментальной кривой лежит точка с номером 1, поэтому таблично заданную функцию будем заменять первой кривой $y = ax^b$ (см. таблицу 2.1).

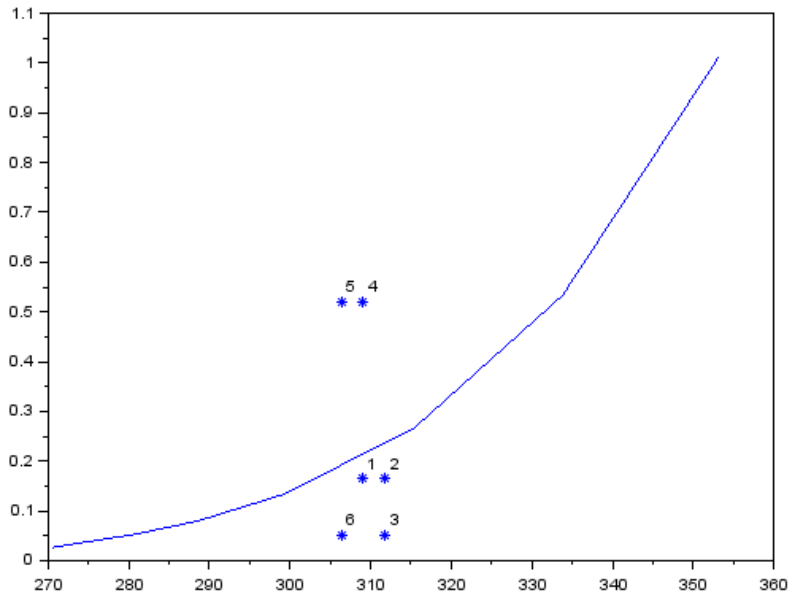


Рис. 2.10. Выбор вида зависимости

Приведем полученную кривую к линейному виду.

Кривую $y = ax^b$ (как и любую другую из таблицы 2.1) приводят к линейному виду (то есть к виду $g = c_0 + c_1 t$) заменой переменных. В нашем случае можно прологарифмировать обе части искомого уравнения: $\ln y = \ln ax^b = \ln a + b \ln x$. Сделаем замену $g=\ln y$, $c_0=\ln a$, $c_1=b$, $t=\ln x$.

Коэффициенты (параметры) c_0 и c_1 находят методом наименьших квадратов, то есть добиваются того, чтобы $K = \sum_{i=0}^{m} (g_i - c_0 - c_1 t_i)^2$ было минимальным. Эта задача сводится к составлению и решению системы нормальных уравнений:

$$\begin{cases} \sum_{i=0}^{m} g_i = (m+1)c_0 + c_1 \sum_{i=0}^{m} t_i \\ \sum_{i=0}^{m} g_i t_i = c_0 \sum_{i=0}^{m} t_i + c_1 \sum_{i=0}^{m} t_i^2 \end{cases} \text{ или } \begin{cases} \sum_{i=0}^{m} \ln y_i = (m+1)c_0 + c_1 \sum_{i=0}^{m} \ln x_i \\ \sum_{i=0}^{m} \ln y_i \ln x_i = c_0 \sum_{i=0}^{m} \ln x_i + c_1 \sum_{i=0}^{m} \ln^2 x_i \end{cases}.$$

Вычислим необходимые суммы в Excel:

	T	x	yi	lny	lnx	lnx*lny	(lnx)^2
	270,5	0,705	0,0267	-3,62309	-0,34956	1,266479	0,12219
	280,8	0,808	0,0533	-2,93182	-0,21319	0,625044	0,045451
	288,6	0,886	0,08	-2,52573	-0,12104	0,30571	0,01465
	299,2	0,992	0,1333	-2,01515	-0,00803	0,016186	6,45E-05
	315,4	1,154	0,2667	-1,32163	0,143234	-0,1893	0,020516
	333,8	1,338	0,5333	-0,62867	0,291176	-0,18305	0,084783
	353,2	1,532	1,0133	0,013212	0,426574	0,005636	0,181965
СУММА				-13,0329	0,169163	1,846698	0,469621

В нашем примере получается система:

$$\begin{cases} -13{,}0329 = 7c_0 + 0{,}169163c_1 \\ 1{,}846698 = 0{,}169163c_0 + 0{,}4696216c_1 \end{cases}$$

Решая ее (например, в Scilab), получаем c_0=-1,9740558; c_1= 4,6433937. Зная c_0 и c_1, находят a и b. В нашем примере

$$b = c_1 = 4{,}6433937; \quad a = e^{c_0} = 0{,}1388924.$$

Таким образом, мы заменили таблично заданную функцию кривой $p = 0{,}1388924 \cdot \left(\frac{T - 200}{100} \right)^{4.6433937}$.

Подставляя в полученную формулу с конкретными значениями a и b значения x_0, x_1, … , x_m и сравнивая их с исходными данными, делают выводы об адекватности математической модели. В рассмотренном примере:

	T	yi	y=0,1388924((T-200)/100)^4,6433937	(yi-y)^2
	270,5	0,0267	0,027400645	4,90903E-07
	280,8	0,0533	0,051612272	2,84843E-06
	288,6	0,08	0,079175818	6,79276E-07
	299,2	0,1333	0,133807607	2,57665E-07
	315,4	0,2667	0,270099974	1,15598E-05
	333,8	0,5333	0,536863654	1,26996E-05
	353,2	1,0133	1,006718075	4,33217E-05
СУММА				7,18575E-05

В нашем случае модель получилась адекватной. Сумма площадей квадратов отклонений равна 0,00007 (последний столбец таблицы). Ниже приведен график полученной кривой (рис. 2.11).

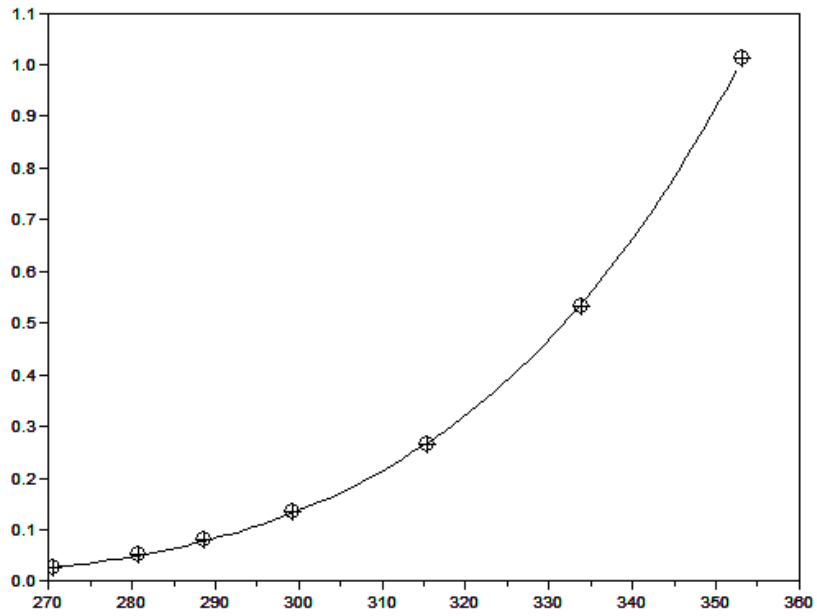


Рис. 2.11. Аппроксимирующая кривая и исходные данные

Решение данной задачи в среде Scilab:

```
clc
clear
clf()
T=[270.5 280.8 288.6 299.2 315.4 333.8 353.2];
y=[0.0267 0.0533 0.08 0.1333 0.2667 0.5333 1.0133];
m=length(T);
x=(T-200)/100;
A=[m sum(log(x));sum(log(x)) sum(log(x).^2)];
e=[sum(log(y)) sum(log(y).*log(x))];//левая часть системы уравнений
d=e/A';a=exp(d(1));b=d(2);t=min(T)-1:0.1:max(T)+1;
Yt=a*((t-200)/100).^b;
plot2d(T,y,-3);
plot2d(t,Yt);S=sum((a*((T-200)/100).^b-y).^2);
disp(S,'S='); disp('a=',a);disp('b=', b)
```

S= 0.0000719

a= 0.1388928

b= 4.6433884

$$p = 0{,}1388928 \cdot \left(\frac{T - 200}{100} \right)^{4.6433884} .$$

Однако надо помнить, что методом наименьших квадратов мы определили коэффициенты прямой $g = c_0 + c_1 t$. Переход от нее к искомой кривой вносит погрешности в результат решения исходной задачи.

Покажем, как можно решить задачу аппроксимации с помощью функции *datafit* в случае, когда аппроксимирующая кривая – не алгебраический полином. Решим пример 2.2 методом средних точек в среде Scilab с использованием функции *datafit*.

Сначала строим график для выбора вида зависимости.

```
clc
clf()
x=[270.5 280.8 288.6 299.2 315.4 333.8 353.2];
y=[0.0267 0.0533 0.08 0.1333 0.2667 0.5333 1.0133];
n=length(x);
xap=(x(1)+x(n))/2
xgar=2*x(1)*x(n)/(x(1)+x(n))
xgeo=(x(1)*x(n))^.5
yap=(y(1)+y(n))/2
ygar=2*y(1)*y(n)/(y(1)+y(n))
ygeo=(y(1)*y(n))^.5
t=[xgeo xap xap xgeo xgar xgar]
z=[ygeo ygeo ygar yap yap ygar]
plot(x,y)
plot(t,z,'*')
```

```
for i=1:6
xnumb(t(i),z(i),i)
end
```

Программа строит график по исходным данным, для каждой из шести кривых таблицы 2.1 подсчитываются координаты средних точек и наносятся на тот же график (см. рис. 2.10 на с. 21). Из графика видно, что искомая кривая – кривая номер 1.

Для уменьшения вычислительной погрешности в программе сделана замена переменной: $x = \dfrac{T - 200}{100}$

```
clc;clf();t1=[270.5 280.8 288.6 299.2 315.4 333.8 353.2];
y=[0.0267 0.0533 0.08 0.1333 0.2667 0.5333 1.0133];
function [zr]=F(c,z),zr=z(2)-(c(1)*z(1)^c(2)),endfunction
x=(t1-200)/100;z=[x;y];c=[1;1];[d,S]=datafit(F,z,c)
t=min(t1):max(t1);Yt=(d(1)*((t-200)/100).^d(2))
plot2d(t1,y,-3);plot2d(t,Yt); Y=(d(1)*x.^d(2));K=sum((y-Y).^2)
disp('K=', K); disp('d=', d);
```

Результаты работы программы:

```
K  =   0.0000288
d  =   0.136781
       4.691623
```

Таким образом, уравнение аппроксимирующей кривой имеет вид

$$p = 0{,}1367810\left(\frac{\mathrm{T} - 200}{100}\right)^{4{,}691623}$$

Сумма площадей квадратов отклонений K=0.0000288.

График этой кривой показан на рис. 2.12.

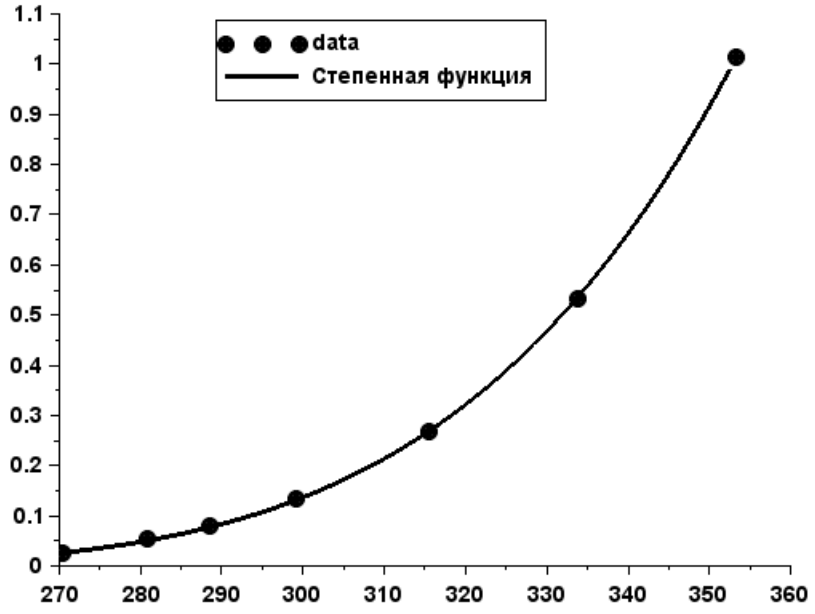


Рис. 2.12. Аппроксимирующая кривая и исходные данные

Решение задачи аппроксимации с помощью функции *datafit* несколько отличается от решения задачи, полученного на с. 24. Там получилась следующая зависимость:

$$p = 0{,}1388928 \cdot \left(\frac{T - 200}{100} \right)^{4.6433884} \cdot$$ Сумма площадей квадратов отклонений в новом решении получилась почти в 2.5 раза меньше. Это связано с тем, что на самом деле коэффициенты искомой кривой $y = ax^b$ методом наименьших квадратов должны быть рассчитаны исходя из условий (2.2) - (2.3), которые, с учетом вида кривой, перепишутся в виде:

$$K = \sum_{i=0}^{m} [y_i - a x_i^b]^2$$

$$K \to \min_{c_j}$$

Именно такая задача решается с помощью функции *datafit*. Решение этой задачи сводится к решению системы двух нелинейных уравнений:

$$\begin{cases} \dfrac{\partial K}{\partial a} = 0, \\ \dfrac{\partial K}{\partial b} = 0. \end{cases}$$

Для нашей кривой эта система выглядит так:

$$\begin{cases} \sum_{i=0}^{m} y_i x_i^b = a \sum_{i=0}^{m} x_i^{2b} \\ \sum_{i=0}^{m} y_i x_i^{b-1} = a \sum_{i=0}^{m} x_i^{2b-1}. \end{cases}$$

Линеаризация задачи (то есть приведение кривой к линейному виду) упрощает решение задачи, но, как правило, точность ее решения падает.

2.1.2. Решение задачи аппроксимации методом наименьших модулей

В качестве меры погрешностей в случае точечной аппроксимации выбирают меру

$$M = \sum_{i=0}^{m} | f(x_i) - fun(x_i, c_j) | \tag{2.10}$$

а коэффициенты c_j ищут из условия

$$M \to \min_{c_j} \tag{2.11}$$

Здесь *fun* – искомая функция, а c_j – искомые коэффициенты. В отличие от МНК минимизируются не суммы квадратов невязок, а суммы (иногда – взвешенные) их абсолютных значений. Напоминаем, что невязка – это раз-

ность между экспериментальными и модельными значениями функции (отрезки прямых зеленого цвета на рис. 2.13).

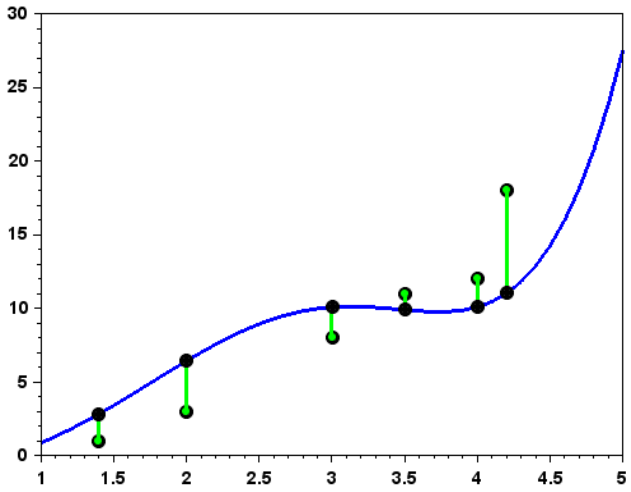


Рис. 2.13. Аппроксимация методом наименьших модулей

Для решения задачи аппроксимации методом наименьших модулей в Scilab можно воспользоваться функцией *fminsearch.* Вот одна из ее разновидностей:

[x,fval,exitflag,output] = fminsearch (costf , x0 , options)

Функция вычисляет минимум заданной целевой функции без ограничений. В этой функции для нахождения минимума используется симплексный метод прямого поиска Нелдера-Мида.

Входными параметрами функции *fminsearch* являются:

costf - целевая функция;

x0 – вектор начальных приближений в случае функции одной переменной или матрица, когда минимизируется функция нескольких переменных;

options – используются для внесения изменений в настройки параметров процесса;

x – точка минимума;

fval – значение целевой функции в точке минимума;

exitflag может принимать следующие значения:

-1 – число итераций превысило максимум;

0 – превышено максимальное число обращений к функции;

1 – получено решение в результате сходимости процесса.

output – структура, содержащая информацию об используемом алгоритме, числе итераций, количестве обращений к функции и сообщение о завершении работы.

Пример 2.3. Пример: построить аппроксимирующую параболу $y = c_0 + c_1 x + c_2 x^2$ для функции, заданной в виде таблицы.

x	0	1	2	3
y	-1,2	-0,1	1,3	1,8

Коэффициенты параболы определить методом наименьших модулей.

Решение. Задачу решим в среде Scilab.

```
clf();clc,clear
t=[0 1 2 3];y=[-1.2 -0.1 1.3 1.8];
function z=G1(x)
  w1=x(1)+x(2)*t+x(3)*t.^2;
dfr=(y-w1);z=sum(abs(dfr));
endfunction
x=[1;1;0]; //Задаем начальное приближение
[a,sse]=fminsearch(G1,x);
plot2d(t,y,-5);m=100;
h=(max(t)-min(t))/m;t1=min(t)-0.1:h:max(t)+0.1;
Ptc1=a(1)+a(2)*t1+a(3)*t1.^2;
 disp('a=',a); disp('sse=',sse)
plot2d(t1,Ptc1,[5]);gca.children.children.thickness=3
gca.font_size = 3;gca.font_style = 8
legend("data","Аппроксимирующая парабола",5)
title('Метод наименьших модулей','fontsize', 5, 'fontname', 3)
```

Запустив программу на выполнение, получаем:

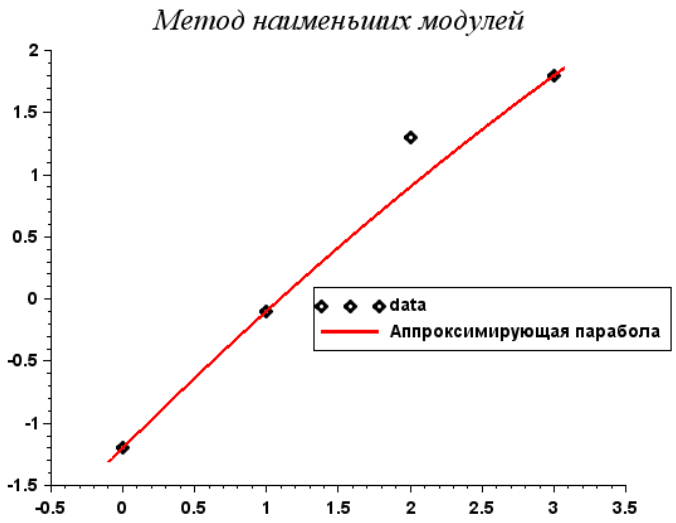


Рис. 2.14. Аппроксимирующая кривая

```
"a1="
 -1.1999993   1.1544546  -0.0514849
"sse="
  0.4000013
```

Уравнение получившейся аппроксимирующей параболы – $y = -1.1999993 + 1.1544546x - 0.0514849x^2$.

x	y	$Y = -1.1999993 + 1.1544546x - 0.0514849x^2$.	|Y-y|
0	-1,2	-1,1999993	7E-07
1	-0,1	-0,0970296	0,0029704
2	1,3	0,9029703	0,3970297
3	1,8	1,8000004	4E-07
sse			0,4000012

Краткий комментарий к программе.

Задаются исходные данные (t, y). Определяется целевая функция *G1(x)*. В ней x(1), x(2), x(3) – искомые коэффициенты, w1 – значения аппроксимирующей параболы в точках t, dfr – вектор невязок. Функция G1 - это сумма модулей невязок. Задается вектор начальных приближений (вектор-столбец x). Входными аргументами функции fminsearch являются функция G1 и вектор x, выходными – коэффициенты искомой зависимости (вектор a1) и sse – сумма модулей невязок. Далее на график наносятся исходные данные, пределы изменения аргумента разбиваются на 100 равноотстоящих частей и строится график полученной функции по 101 точке. На экран выводятся значение sse и значения искомых коэффициентов (a1). Изменяются толщина кривой, стиль и размер шрифта. Выводится заголовок и легенда (рис. 2.14).

Следует отметить, что точность решения зависит от того, насколько удачно задано начальное приближение.

2.1.3. Равномерное приближение. Решение задачи в среде Scilab

В качестве меры погрешности $\mu(f,Q)$ выбирают

$$R = \max_i \left| f(x_i) - fun(x_i, c_j) \right| \tag{2.12}$$

Коэффициенты c_j подбирают таким образом, чтобы R было минимальным:

$$\min_{c_j} \max_i \left| f(x_i) - fun(x_i, c_j) \right| \tag{2.13}$$

Это есть минимаксный подход, который осуществляет равномерное приближение функций. Геометрически это означает: из всех кривых заданного вида выбирают ту, у которой максимальная по модулю невязка (длина отрезка AB на рис. 2.15) минимальна.

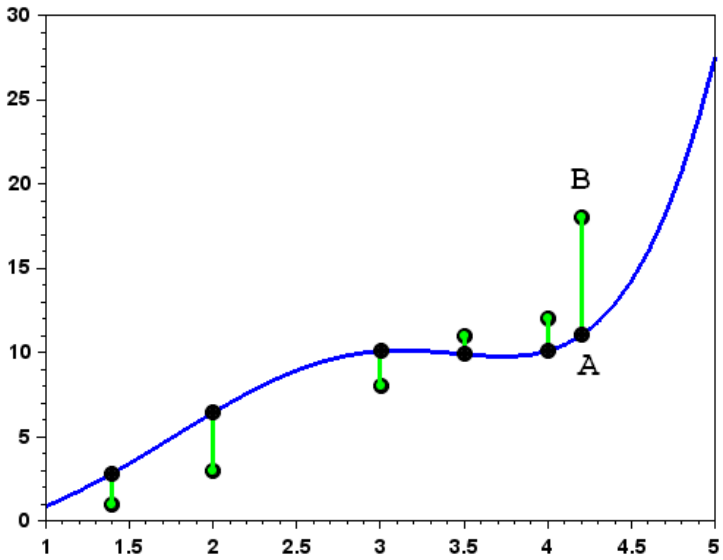


Рис. 2.15. Минимаксный подход к решению задачи аппроксимации

Для решения задачи точечной аппроксимации в этом случае также можно воспользоваться функцией *fminsearch*.

Пример 2.4. Пример: построить аппроксимирующую параболу $y = c_0 + c_1 x + c_2 x^2$ для функции, заданной в виде таблицы.

x	0	1	2	3
y	-1,2	-0,1	1,3	1,8

Коэффициенты параболы определить, используя минимаксный подход.

Решение. Задачу решим в среде Scilab.

```
//МЕТОД МИНИМАКСА
clf();clc,clear
t=[0 1 2 3];y=[-1.2 -0.1 1.3 1.8];
function z=G1(x)
  w1=x(1)+x(2)*t+x(3)*t.^2;
dfr=(y-w1);
z=max(abs(dfr));
endfunction
x=[0;0;0]; //Задаем начальное приближение
[a,sse]=fminsearch(G1,x);
plot2d(t,y,-5);m=100;
h=(max(t)-min(t))/m;
t1=min(t)-0.1:h:max(t)+0.1;
Ptc1= a(1)+a(2)*t1+a(3)*t1.^2;
disp('er=',sse);
disp('a=',a);
plot2d(t1,Ptc1,[5])
gca.children.children.thickness=3
gca.font_size = 3;gca.font_style = 8
legend("data","Аппроксимирующая кривая",5)
title('Метод минимакса','fontsize', 5, 'fontname', 3)
"er="
  0.1579018
 "a="
 -1.3578681   1.5808011  -0.165409
```

Таким образом, уравнение аппроксимирующей кривой

$y=-1.3578681 + 1.5808011x - 0.165409x^2$

x	y	$y=-1.3578681 + 1.5808011x - 0.165409x^2$	|Y-y|
0	-1,2	-1,3578681	0,1578681
1	-0,1	0,057524	0,157524
2	1,3	1,1420981	0,1579019
3	1,8	1,8958542	0,0958542
sse			0,1579019

Исходные данные и полученная кривая представлены на рис. 2.16.

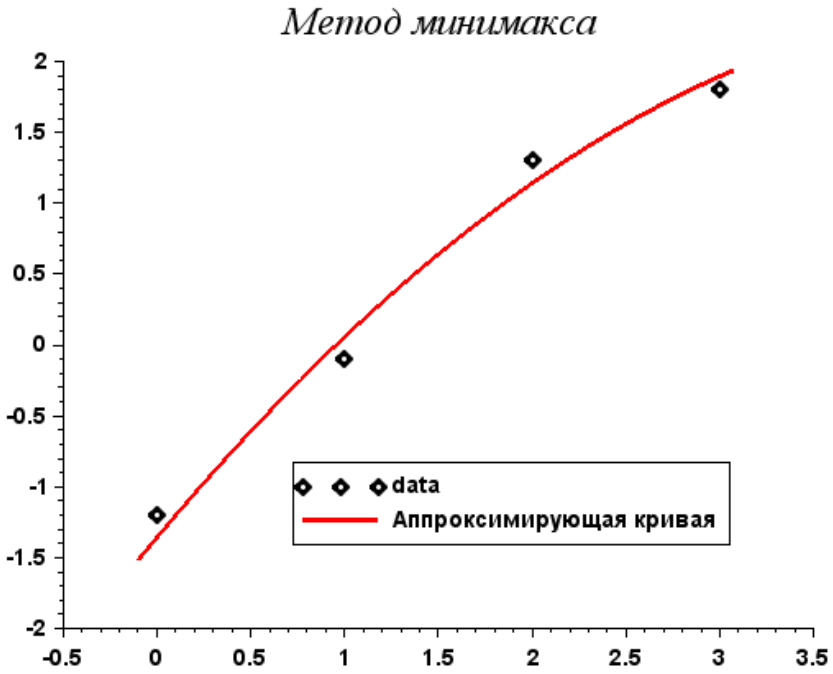


Рис. 2.14. Аппроксимирующая кривая. Коэффициенты полинома получены методом минимакса

При решении задачи аппроксимации по одним и тем же исходным данным мы получили, что и метод наименьших квадратов, и равномерное приближение, и метод наименьших модулей дают приблизительно одинаковые результаты, однако так бывает далеко не всегда.

Заметим, что аппроксимирование экспериментальных данных с помощью функции *datafit* не всегда дает правильные результаты. Полученные решения необходимо проверять. Кроме того, существует множество функций, найти коэффициенты которых методом наименьших квадратов с помощью функции *datafit* нельзя, особенно если аппроксимирующая кривая не является алгебраическим полиномом. Часто полученные решения далеки от реальных.

Иногда, если полученное решение нас не устраивает, правильное решение можно получить, поварьировав значение начального приближения (вектор ***c*** в приведенных программах). Заметим, что и решение задачи аппроксимации МНК может быть получено с помощью функции fminsearch.

Тема 3. Интерполяция

Интерполяция – частный случай аппроксимации, когда аппроксимирующая кривая проходит через все точки таблицы, то есть

$$Q(x_i, c_j) = f(x_i) \quad , \quad i = \overline{0,m} \tag{3.1}$$

Задача интерполяции ставится так:

Функция y=*f*(x) задана в виде таблицы

x	x_0	x_1	…	x_m
y	y_0	y_1	…	y_m

Требуется вычислить значение функции в точке *x*, не совпадающей ни с одним из узлов таблицы. Если эта точка лежит между узлами, то такая задача называется интерполяцией. Если точка *x* лежит за пределами таблицы, то говорят о задаче экстраполяции.

Очевидно, что коэффициенты c_j можно найти методом наименьших квадратов, положив *n*=*m*.

Рассмотрим другие способы решения этой задачи.

3.1 Интерполяция по Лагранжу

Построим полином

$$L_n(x) = \sum_{i=0}^{n} y_i \prod_{j \neq i} \frac{x - x_j}{x_i - x_j} \tag{3.2}$$

Выражение (3.2) – алгебраический полином степени *n*. Покажем, что он проходит через все точки таблицы при *n*=2 и *n*=3.

$$L_2(x) = y_0 \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} + y_1 \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} + y_2 \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}$$

$$L_2(x_0) = y_0 + 0 + 0;\ L_2(x_1) = 0 + y_1 + 0;\ L_2(x_2) = 0 + 0 + y_2;$$

$$L_3(x) = y_0 \frac{(x-x_1)(x-x_2)(x-x_3)}{(x_0-x_1)(x_0-x_2)(x_0-x_3)} + y_1 \frac{(x-x_0)(x-x_2)(x-x_3)}{(x_1-x_0)(x_1-x_2)(x_1-x_3)} +$$

$$+ y_2 \frac{(x-x_0)(x-x_1)(x-x_3)}{(x_2-x_0)(x_2-x_1)(x_2-x_3)} + y_3 \frac{(x-x_0)(x-x_1)(x-x_2)}{(x_3-x_0)(x_3-x_1)(x_3-x_2)}$$

$$L_3(x_0) = y_0, \quad L_3(x_1) = y_1, L_3(x_2) = y_2, L_3(x_3) = y_3$$

Выражение (3.2) – интерполяционный полином Лагранжа, проходящий через все точки таблицы. Теперь, чтобы найти значение функции в некоторой точке x^*, которой нет в таблице, достаточно подставить эту точку вместо *x* в формулу (3.2) и считать, что

$$y^* = L_n(x^*) \tag{3.3}$$

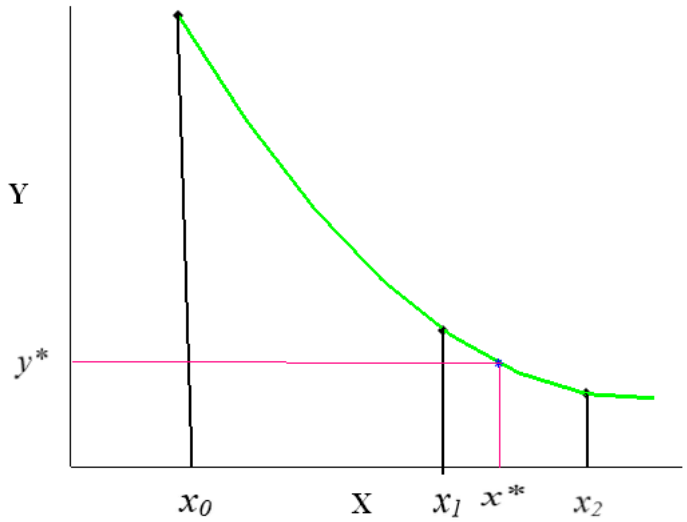


Рис. 3.1. Графическое решение задачи интерполяции

Пример 3.1. Зависимость вязкости (η) нитробензола от температуры выражается следующими экспериментальными данными:

T, K	293.15	303.15	313.15
η, мПа·с	2.013	1.682	1.438

Определить вязкость нитробензола при температуре *T=301* К.

Решение. Пусть *x* – это температура, *y* – вязкость. Поскольку число узлов интерполяции равно трем, воспользуемся формулой интерполяционного полинома второй степени.

$$y = L_2(x) = 2.013\frac{(x-303.15)(x-313.15)}{(293.15-303.15)(293.15-313.15)} + 1.682\frac{(x-293.15)(x-313.15)}{(303.15-293.15)(303.15-313.15)} +$$

$$+1.438\frac{(x-293.15)(x-303.15)}{(313.15-293.15)(313.15-303.15)} = 50.374029 - 0.2924905x + 0.000435x^2$$

Подставив в формулу заданное значение температуры, получим, что вязкость нитробензола при температуре *T*=301 К равна: $y = L_2(301) = 1.7458233$ мПа·с.

3.2. Интерполяция при постоянном шаге

Конечной разностью функции *y=f*(*x*) с шагом Δ*x=h* называют функцию Δ*y=f*(*x*+*h*)-*f*(*x*). Это первая конечная разность. Вторая конечная разность $\Delta^2 y=[f(x+2h)-f(x+h)]-[f(x+h)-f(x)]=f(x+2h)-2f(x+h)+f(x)$.

Предположим, что функция *y=f*(*x*) задана в виде таблицы из четырех точек. Построим для нее таблицу конечных разностей:

i	x_i	y_i	Δy_i	$\Delta^2 y_i$	$\Delta^3 y_i$
0	x_0	y_0	Δy_0	$\Delta^2 y_0$	$\Delta^3 y_0$
1	x_1	y_1	Δy_1	$\Delta^2 y_1$	
2	x_2	y_2	Δy_2		
3	x_3	y_3			

В этой таблице

$\Delta y_0 = y_1 - y_0; \Delta y_1 = y_2 - y_1; \Delta y_2 = y_3 - y_2; \quad \Delta^2 y_0 = \Delta y_1 - \Delta y_0;$

$\Delta^2 y_1 = \Delta y_2 - \Delta y_1; \quad \Delta^3 y_0 = \Delta^2 y_1 - \Delta^2 y_0.$

Предположим, что шаг в этой таблице – постоянный, то есть

$$x_{i+1} - x_i = h_i = h = const \quad i = \overline{0,3}. \tag{3.4}$$

Запишем полином, проходящий через все точки таблицы (это будет полином степени не выше третьей), в следующем виде:

$$Q_3(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + a_3(x - x_0)(x - x_1)(x - x_2) \tag{3.5}$$

Имеем: $Q_3(x_0) = y_0 \quad \Rightarrow \quad Q_3(x_0) = a_0 + 0$; т.е. $a_0 = y_0$

$$Q_3(x_1) = y_1 \quad \Rightarrow \quad Q_3(x_1) = y_1 = y_0 + a_1(x_1 - x_0) + 0; \quad a_1 = \frac{y_1 - y_0}{x_1 - x_0} = \frac{\Delta y_0}{h}$$

$$Q_3(x_2) = y_2 \quad \Rightarrow \quad Q_3(x_2) = y_2 = y_0 + a_1(x_2 - x_0) + a_2(x_2 - x_0)(x_2 - x_1) + 0;$$

$$a_1 = \frac{y_1 - y_0}{x_1 - x_0} = \frac{\Delta y_0}{h}; \quad y_2 = y_0 + \frac{\Delta y_0}{h} 2h + a_2 \cdot 2h \cdot h; \quad a_2 = \frac{y_2 - y_0 - 2y_1 + 2y_0}{2h^2} = \frac{\Delta^2 y_0}{2!h^2}.$$

Аналогично, $a_3 = \frac{\Delta^3 y_0}{3!h^3}$. Обозначим $q_1 = \frac{x - x_0}{h}$. После подстановки в (3.5) значений a_i, $\quad i = \overline{0,3}$ и q_1 окончательно получаем:

$$Q_3(x) = y_0 + q_1 \Delta y_0 + \frac{q_1(q_1 - 1)}{2!} \Delta^2 y_0 + \frac{q_1(q_1 - 1)(q_1 - 2)}{3!} \Delta^3 y_0$$

Запишем общий вид полинома *n*-й степени:

$$Q_n^I(x) = y_0 + q_1 \Delta y_0 + \frac{q_1(q_1 - 1)}{2!} \Delta^2 y_0 + \frac{q_1(q_1 - 1)(q_1 - 2)}{3!} \Delta^3 y_0 + \frac{q_1(q_1 - 1)(q_1 - 2)(q_1 - 3)}{4!} \Delta^4 y_0 +$$

$$+ \ldots + \frac{q_1(q_1 - 1)(q_1 - 2)\ldots(q_1 - n + 1)}{n!} \Delta^n y_0 \tag{3.6}$$

«*I*» означает, что это первый интерполяционный полином Ньютона. Он удобен для вычислений, когда точка x^*, в которой нужно вычислить значение функции, расположена ближе к началу таблицы.

Если точка x^* расположена ближе к концу таблицы, то удобнее пользоваться формулой второго интерполяционного полинома Ньютона:

$$Q_n^{II}(x) = y_n + q_2\Delta y_{n-1} + \frac{q_2(q_2+1)}{2!}\Delta^2 y_{n-2} + \frac{q_2(q_2+1)(q_2+2)}{3!}\Delta^3 y_{n-3} + \ldots$$

$$+ \frac{q_2(q_2+1)(q_2+2)\ldots(q_2+n-1)}{n!}\Delta^n y_0 \quad \text{, где } q_2 = \frac{x - x_n}{h} \qquad (3.7)$$

Результаты применения формул (3.6) и (3.7) – одни и те же, если используются одни и те же узлы таблицы.

Пример 3.2. Зависимость теплоемкости этана C_p от температуры T задана таблицей:

T, К	498	598	698	798	898
C_p, Дж/(моль·К)	78.03	89.02	98.91	107.76	115.62

Определить, какова теплоемкость этана при T=870 К и T=522 К.

Решение. Так как шаг в таблице – постоянный (h=100), для решения задачи можно воспользоваться формулой интерполяционного полинома Ньютона. Построим таблицу конечных разностей. В ней $x=T$, $y=C_p$.

x	y	Δy_0	$\Delta^2 y_0$	$\Delta^3 y_0$	$\Delta^4 y_0$
498	78.03	10.99	-1.1	0.06	-0.01
598	89.02	9.89	-1.04	0.05	
698	98.91	8.85	-0.99		
798	107.76	7.86			
898	115.62				

Из таблицы видно, что конечная разность четвертого порядка отлична от нуля, поэтому будем строить интерполяционный полином четвертой степени.

а) Так как точка x = 870 расположена ближе к концу таблицы, воспользуемся формулой второго интерполяционного полинома Ньютона (3.7), приняв x_n= 898; y_n=115.62;

$h = x_{i+1} - x_i = 100$. Тогда $q_2 = \frac{870 - 898}{100} = -0.28.$

Подставляя исходные данные в формулу (3.7), получим:

$$Q_4^{II}(870) = 115.62 + (-0.28)\cdot 7.86 + \frac{(-0.28)(-0.28+1)}{2}\cdot(-0.99) +$$

$$+ \frac{(-0.28)(-0.28+1)(-0.28+2)}{6}\cdot 0.05 + \frac{(-0.28)(-0.28+1)(-0.28+2)(-0.28+3)}{24}\cdot(-0.01) =$$

$$=113.5165.$$

Теплоемкость этана при T = 870 К равна 113.5165 Дж/(моль·К).

b) Так как точка x=522 расположена ближе к началу таблицы, воспользуемся формулой первого интерполяционного полинома Ньютона (3.6), приняв x_0= 498; y_0=78.03.

Тогда $q_1 = \dfrac{522-498}{100} = 0.24.$

Подставляя исходные данные в формулу (2.6), получим:

$$Q_4^I(522) = 78.03 + 0.24 \cdot 10.99 + \frac{(0.24)(0.24-1)}{2} \cdot (-1.1) + \frac{(0.24)(0.24-1)(0.24-2)}{6} \cdot 0.06 +$$

$$+ \frac{(0.24)(0.24-1)(0.24-2)(0.24-3)}{24} \cdot (-0,01) = 80.7715.$$

Теплоемкость этана при T = 522 К равна 80.7715 Дж/(моль·К).

3.3. Обратная интерполяция

Постановка задачи: функция y задана в виде таблицы

x	x_0	x_1	…	x_m
y	y_0	y_1	…	y_m

Требуется найти значение аргумента x^*, при котором функция принимает некоторое заданное значение y^*.

Задачу можно решить по крайней мере двумя различными способами.

1 способ.

a) Строим интерполяционный полином (как правило, интерполяционный полином Лагранжа) $y = L_n(x)$.

b) Решаем уравнение $y^* = L_n(x)$ любым подходящим численным методом (например, методом деления отрезка пополам).

На рис. 3.2 приведено графическое решение задачи обратной интерполяции первым способом.

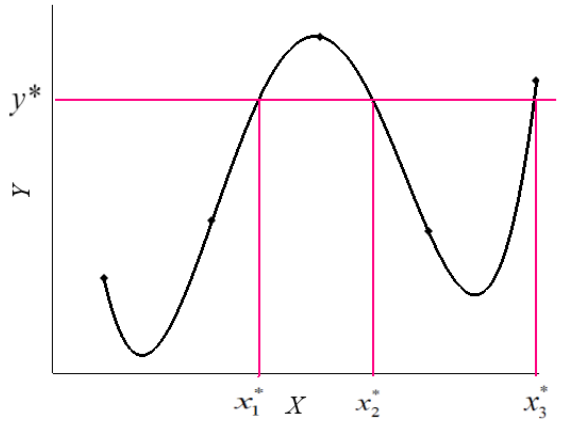


Рис. 3.2. Решение задачи обратной интерполяции первым способом

Из рисунка видно, что задача может иметь не единственное решение.

2 способ.

a) Строим интерполяционный полином $x = L_n(y)$.

b) Подставляем в этот полином значение y^* и получаем x^*: $x^* = L_n(y^*)$.

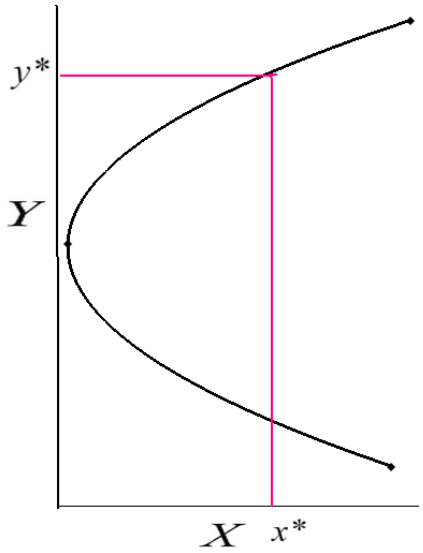


Рис. 3.3. Решение задачи обратной интерполяции вторым способом

На рис. 3.3 приведен график интерполяционного полинома и решение задачи обратной интерполяции вторым способом.

Как правило, решения задачи, полученные первым и вторым способом, будут различными.

Пример 3.3. Данные по плотности ρ водных растворов хлорида магния в зависимости от его концентрации C при температуре 293 К приведены в таблице:

C, %	2.00	4.00	8.00	16.00
ρ, г/см^3	1.0146	1.0311	1.0646	1.1342

Определить, при какой концентрации плотность раствора хлорида магния будет равна 1.1031 г/см^3 [5].

Решение. Обозначим $x=C$, $y=\rho$.

Первый способ:

Строим полином

$$y = L_3(x) = y_0 \frac{(x-x_1)(x-x_2)(x-x_3)}{(x_0-x_1)(x_0-x_2)(x_0-x_3)} + y_1 \frac{(x-x_0)(x-x_2)(x-x_3)}{(x_1-x_0)(x_1-x_2)(x_1-x_3)} +$$

$$+ y_2 \frac{(x-x_0)(x-x_1)(x-x_3)}{(x_2-x_0)(x_2-x_1)(x_2-x_3)} + y_3 \frac{(x-x_0)(x-x_1)(x-x_2)}{(x_3-x_0)(x_3-x_1)(x_3-x_2)}$$

$$y = L_3(x) = 1.0146\frac{(x-4)(x-8)(x-16)}{(2-4)(2-8)(2-16)} + 1.0311\frac{(x-2)(x-8)(x-16)}{(4-2)(4-8)(4-16)} +$$

$$+1.0646\frac{(x-2)(x-4)(x-16)}{(8-2)(8-4)(8-16)} + 1.1342\frac{(x-2)(x-4)(x-8)}{(16-2)(16-4)(16-8)} =$$

$$= 0.0000004\,x^3 \;+\; 0.0000146\,x^2 + \;0.00815x \;\;+\;\; 0.9982381.$$

Решаем уравнение

$$0.0000004\;x^3 \;+\; 0.0000146x^2 + \;0.00815\,x \;\;+\;\; 0.9982381 = 1.1031$$

любым численным методом. В данном примере решение было найдено с использованием функции *roots* среды Scilab. Уравнение имеет единственный действительный корень x=12.481237. Таким образом, ρ=1.1031 при $\boldsymbol{C}$=12.48 %.

Второй способ:

Строим полином третьей степени

$$x = L_3(y) = x_0\frac{(y-y_1)(y-y_2)(y-y_3)}{(y_0-y_1)(y_0-y_2)(y_0-y_3)} + x_1\frac{(y-y_0)(y-y_2)(y-y_3)}{(y_1-y_0)(y_1-y_2)(y_1-y_3)} +$$

$$+ x_2\frac{(y-y_0)(y-y_1)(y-y_3)}{(y_2-y_0)(y_2-y_1)(y_2-y_3)} + x_3\frac{(y-y_0)(y-y_1)(y-y_2)}{(y_3-y_0)(y_3-y_1)(y_3-y_2)}$$

Подставляем данные*:*

$$x = L_3(y) = 2\frac{(y-1.0311)(y-1.0646)(y-1.1342)}{(1.0146-1.0311)(1.0146-1.0646)(1.0146-1.1342)} +$$

$$+4\frac{(y-1.0146)(y-1.0646)(y-1.1342)}{(1.0311-1.0146)(1.0311-1.0646)(1.0311-1.1342)} +$$

$$+8\frac{(y-1.0146)(y-1.0311)(y-1.1342)}{(1.0646-1.0146)(1.0646-1.0311)(1.0646-1.1342)} +$$

$$+16\frac{(y-1.0146)(y-1.0311)(y-1.0646)}{(1.1342-1.0146)(1.1342-1.0311)(1.1342-1.0646)} =$$

$$= -92.898594 + 4.3618469\;y + 147.95491\;y^2 - 59.202545\;y^3$$

При y=1,1031 получаем: $x = L_3(1.1031) = 12.482202$

Таким образом, ρ=1.1031 при $\boldsymbol{C}$=12.48%.

В данном примере ответы, полученные первым и вторым способом, совпали. Конечно, раскрывать скобки и приводить подобные члены крайне затруднительно, поэтому конечный результат – нахождение коэффициентов полинома – получен в среде Scilab (см. с. 49). Коэффициенты полинома можно получить и в Excel (см. см. с. 14).

Примечание: Несколько слов о точности интерполяции. Если исходная функция является полиномом степени *n*, то имеет место тождественное совпадение: $Q(x,c_j) = f(x)$. В противном случае их значения совпадают только в узловых точках, а вне узлов таблицы $R(x) = Q(x,c_j) - f(x) \neq 0$. Эта разность – погрешность интерполяции – называется остаточным членом интерполяционной формулы [1-3]. Если исходная функция непрерывна и имеет непрерывные производные до *n*+1 включительно, то остаточный член интерполяционного полинома Лагранжа имеет вид:

$$R_L(x) = \frac{(x-x_0)(x-x_1)...(x-x_n)}{(n+1)!} f^{(n+1)}(x^*)$$

а остаточный член интерполяционного полинома Ньютона

$$R_Q(x) = \frac{q(q-1)...(q-n)}{(n+1)!} f^{(n+1)}(x^{**})h^{n+1},\ q = \frac{x-x_0}{h}.$$

Здесь $f^{(n+1)}(x)$ – производная порядка *n*+1, вычисленная в некоторой точке отрезка [x_0 , x_n]. Если максимальное значение этой производной на указанном отрезке равно M_{n+1}, то можно оценить остаточный член каждой формулы. Так,

$$\left|R_L(x)\right| \le \left|\frac{(x-x_0)(x-x_1)...(x-x_n)}{(n+1)!} M_{n+1}\right|$$

Поясним вышесказанное следующим примером. Предположим, что функция, которую мы хотим заменить интерполяционным полиномом, задана явно. Пусть, например, эта функция *y=sin(x)+sin(2x).* Отрезок, на котором будем проводить интерполяцию – [1, 10]. Вычислим значение исходной функции в точках x=1,2, …, 10. Построим на отрезке [1, 10] график двух функций: заданной функции *y=sin(x)+sin(2x)* и интерполяционного полинома 9 степени, проходящего через 10 полученных в результате табулирования точек (рис. 3.4).

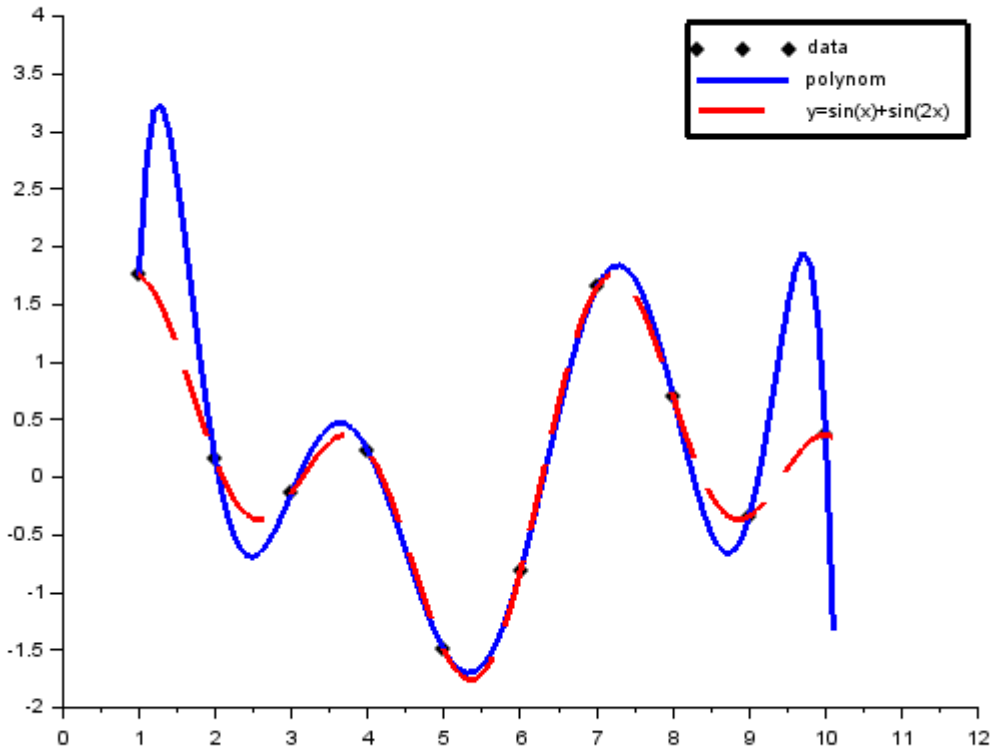


Рис. 3.4. Интерполяция полиномом 9 степени

Вот программа построения интерполяционного полинома.

```
clc
x=1:10;
function y=f(x)
y=sin(x)+sin(2*x);
endfunction
n=length(x);scf(2);clf()
n=length(x);a=[];b=[];c=[];
for i=1:n
  for j=1:n
    a(i,j)=sum(x.^(i+j-2));
  end
  b(i)=sum(x.^(i-1).*f(x));
end
c=inv(a)*b
function z=f1(t)
  z=0;
  for i=1:n
 z=z+t.^(i-1).*c(i);
```

```
  end
  endfunction
plot2d(x,f1(x),-4)
t=min(x):0.1:max(x)+.1;plot(t,f1(t),t,f(t),'--r')
legend('data', 'polynom', 'y=sin(x)+sin(2x)')
err=sum((f(x)-f1(x)).^2)
aa=gca();
aa.data_bounds=[0.5 -2;10.5 3.5];
gca.children.children.thickness=3
```

Видим, что в начале и в конце отрезка интерполяции значения функций вне узловых точек сильно отличаются друг от друга. Интерполяция сплайнами, рассмотренная в следующем разделе, в некоторых случаях помогает устранить этот недостаток.

3.4. Интерполяция сплайнами

Если число узлов интерполяции велико, то построение интерполяционного полинома, проходящего через все точки таблицы, не всегда целесообразно [2]. Для уменьшения погрешности часто поступают следующим образом. На каждом из участков $[x_i, x_{i+1}]$ функцию заменяют сплайном. *Сплайном* называется кусочно-полиномиальная функция, определенная на отрезке $[x_0, x_n]$ и имеющая на этом отрезке некоторое количество непрерывных производных. Преимущества интерполяции сплайнами по сравнению с обычными методами интерполяции – в сходимости и устойчивости вычислительного процесса.

Чаще всего ограничиваются кубическими сплайнами. Всюду далее будем предполагать, что точки x_i упорядочены по возрастанию: $x_0 < x_1 < x_2 < x_3 < \ldots < x_n$.

Запишем уравнение кубического сплайна, проходящего через два соседних узла интерполяции, в следующем виде:

$$S_i(x) = a_i + b_i(x - x_{i-1}) + c_i(x - x_{i-1})^2 + d_i(x - x_{i-1})^3,\ i = \overline{1, n} \qquad (3.8)$$

Здесь $S_1(x)$ - сплайн, проходящий через точки $[x_0, x_1]$, $S_n(x)$ – сплайн, проходящий через точки $[x_{n-1}, x_n]$. Заметим, что $S_i(x_{i-1}) = a_i$.

Для того, чтобы найти коэффициенты всех сплайнов (а их будет n: для трех точек – 2 и т.д.), необходимо получить $4n$ уравнений. Так как каждый сплайн проходит через два узла таблицы, имеем $2n$ уравнений:

$$S_1(x_0) = y_0$$
$$S_1(x_1) = y_1$$
$$S_2(x_1) = y_1$$
$$S_2(x_2) = y_2$$
$$\ldots\ldots\ldots\ldots$$
$$S_n(x_{n-1}) = y_{n-1}$$
$$S_n(x_n) = y_n$$

Еще $2n$-2 уравнения можно получить, приравнивая значения первых и вторых производных соседних сплайнов во внутренних узлах.

$$S_1'(x_1) = S_2'(x_1)$$
$$S_2'(x_2) = S_3'(x_2)$$
$$S_3'(x_3) = S_4'(x_3)$$
$$\ldots\ldots\ldots\ldots\ldots$$
$$S_{n-1}'(x_{n-1}) = S_n'(x_{n-1})$$
$$.S_1''(x_1) = S_2''(x_1)$$
$$S_2''(x_2) = S_3''(x_2)$$
$$S_3''(x_3) = S_4''(x_3)$$
$$\ldots\ldots\ldots\ldots\ldots$$
$$S_{n-1}''(x_{n-1}) = S_n''(x_{n-1})$$

Два последних уравнения для поиска коэффициентов a_i, b_i, c_i, d_i можно получить, задавая значения первой и/или второй производной в точке x_0 или x_n. Например, можно положить

$$S_1''(x_0) = 0,\ S_n''(x_n) = 0. \qquad (3.9)$$

Пример 3.4. Для функции, заданной в виде таблицы, построить два кубических сплайна: на отрезке [1,2] и на отрезке [2, 4].

x	1	2	4
y	8	12	6

Решение. Обозначим первый сплайн $S_1(x)=a+b\cdot x+c\cdot x^2+d\cdot x^3$, второй – $S_2(x)=a_1+b_1\cdot x+c_1\cdot x^2+d_1\cdot x^3$. Первый сплайн проходит через две точки: (x_0, y_0) и (x_1, y_1), второй – через точки (x_1, y_1) и (x_2, y_2).

Имеем 4 уравнения: $S_1(x_0)=y_0,\ S_1(x_1)=y_1;\ S_2(x_1)=y_1,\ S_2(x_2)=y_2$.

Вычислим первые и вторые производные от каждого сплайна и приравняем их значения в точке x_1. Получим еще 2 уравнения.

$S_1'(x) = b + 2cx + 3dx^2$, $S_2'(x) = b_1 + 2c_1x + 3d_1x^2$, $S_1''(x) = 2c + 6dx$, $S_2''(x) = 2c_1 + 6d_1x$.
$S_1'(x_1) = S_2'(x_1)$, $S_1''(x_1) = S_2''(x_1)$.

Последние два уравнения можно получить, например, приравняв к нулю первые производные в первой и последней точках таблицы: $S_1'(x_0) = 0, S_2'(x_2) = 0.$

В итоге получаем систему из восьми уравнений с 8 неизвестными. В соответствии с введенными обозначениями получилась система:

$$a + bx_0 + cx_0^2 + dx_0^3 = y_0$$
$$a + bx_1 + cx_1^2 + dx_1^3 = y_1$$
$$a_1 + b_1x_1 + c_1x_1^2 + d_1x_1^3 = y_1$$
$$a_1 + b_1x_2 + c_1x_2^2 + d_1x_2^3 = y_2$$
$$b + 2cx_1 + 3dx_1^2 - b_1 - 2c_1x_1 - 3d_1x_1^2 = 0$$
$$2c + 6dx_1 - 2c_1 - 6d_1x_1 = 0$$
$$b + 2cx_0 + 3dx_0^2 = 0$$
$$b_1 + 2c_1x_2 + 3d_1x_2^2 = 0$$

Подставляя в систему уравнений исходные данные, получаем:

$$a+b+c+d=8$$
$$a+2b+4c+8d=12$$
$$a_1+2b_1+4c_1+8d_1=12$$
$$a_1+4b_1+16c_1+64d_1=6$$
$$b+4c+12d-b_1-4c_1-12d_1=0$$
$$2c+12d-2c_1-12d_1=0$$
$$b+2c+3d=0$$
$$b_1+8c_1+48d_1=0$$

Решение этой системы линейных уравнений дает значения искомых коэффициентов сплайнов. Решить эту систему уравнений можно, например, в Scilab или Excel.

В результате решения этой системы получили два сплайна:

$S_1(x)=23-35.5x+26x^2-5.5x^3$

$S_2(x)=-38+56x-19.75x^2+2.125x^3$

Напоминаем, что каждый из сплайнов проходит через две соседние точки таблицы, в точке «стыка» (то есть в точке x_1) значения первых и вторых производных от каждого сплайна совпадают, первая производная от первого сплайна в точке x_0 и первая производ-

ная от второго сплайна в точке x_2 равны нулю. Графики этих сплайнов представлены на рис. 3.5.

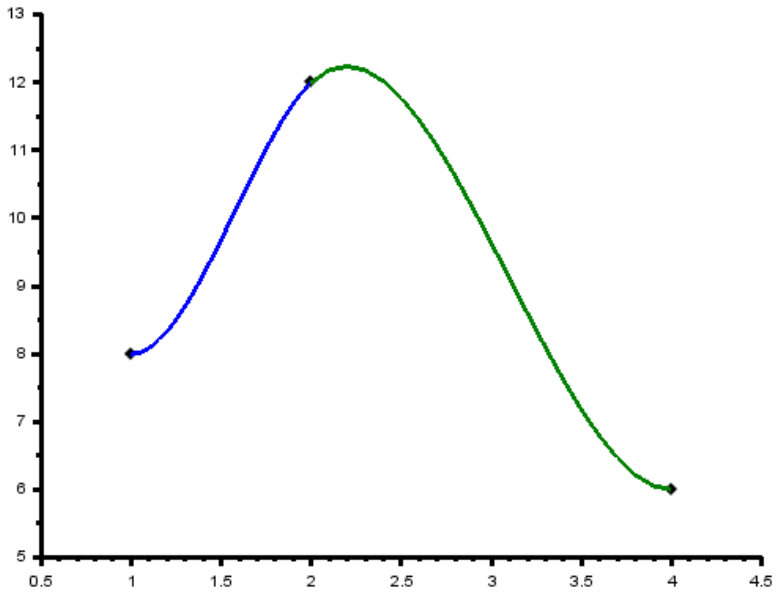


Рис. 3.5. Графики двух полученных кубических сплайнов

3.5. Решение задачи интерполяции в Scilab

3.5.1. Построение интерполяционных полиномов

3.5.1.1. Построение таблицы конечных разностей

Покажем, как в Scilab можно построить таблицу конечных разностей, используемую для вычислений с помощью интерполяционных полиномов Ньютона. Для примера 3.2 это можно сделать так.

```
clc;y=[78.03 89.02 98.91 107.76 115.62];
dy=diff(y),dy2=diff(y,2),dy3=diff(y,3),dy4=diff(y,4)
disp('y=',y,'dy2=',dy2,'dy3=',dy3,'dy4=',dy4)
```

Результаты работы программы:

```
"y="   78.03   89.02   98.91   107.76   115.62
 "dy="   10.99   9.89   8.85   7.86
 "dy2="  -1.1  -1.04  -0.99
 "dy3=   0.06   0.05
 "dy4="  -0.0100000
```

3.5.1.2. Линейная интерполяция в Scilab. Функции interpln и linear_interpn

Если заданы значения функции в двух точках (x_i, y_i) и (x_{i+1},y_{i+1}), то используя линейную интерполяцию значение функции в точке x^* вычисляют по формуле $y = y_i + \frac{(y_{i+1} - y_i)}{(x_{i+1} - x_i)}(x^* - x_i)$. Для линейной интерполяции в Scilab применяются функции interpln и linear_interpn.

Синтаксис функции interpln:

[y1]=interpln(xyd,x1)

Аргументами функции являются: xyd – матрица, состоящая из двух строк – координат векторов одинаковой размерности *x* и *y*. *x* – значения аргумента, *y* – значения функции. Координаты вектора *x* должны быть проранжированы по возрастанию. x1 – вектор, содержащий точки, в которых нужно вычислить значения функции. Значения функции y1 вычисляются при помощи линейной интерполяции. Значения функции за пределами таблицы, то есть при *x*<xmin и *x*>xmax (задача экстраполяции), вычисляют по двум первым и двум последним точкам заданной таблицы соответственно.

Пример 3.5. Функция задана в виде таблицы из пяти точек

x	1	10	20	30	40
y	1	30	-10	20	40

Нанести на график исходные данные и ломаную, полученную линейной интерполяцией. С помощью линейной интерполяции вычислить значения функции в точках -1.2, 23.5 и 41.4.

```
clf();clear;x=[1 10 20 30 40];y=[1 30 -10 20 40];
plot2d(x',y,[-3],"011"," ",[-10,-30,50,50]);
xx=[-1.2 23.5 41.4];y1=interpln([x;y],-4:45);y2=interpln([x;y],xx);
disp(" x        y");disp([xx;y2]')
plot2d((-4:45)',y1',[5],"000");gca.children.children.thickness=3
legend("Точки интерполяции","Интерполяционная кривая",2)
gca.thickness=3
```

Результат работы программы:

```
" x        y"
  -1.2   -6.0888889
   23.5   0.5
   41.4   42.8
```

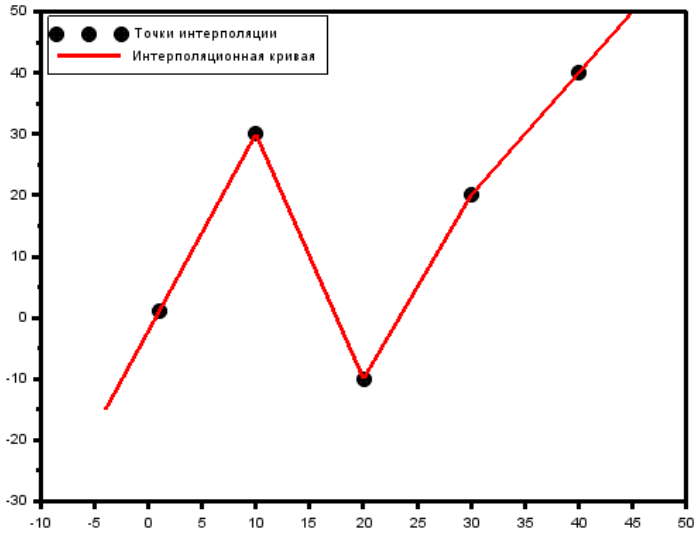


Рис. 3.6. Линейная интерполяция с помощью функции interpln

В приведенной программе установлены границы рамки графика: x (от -10 до 50) и y (от -30 до 50). Сам график функции на рис. 3.6 построен на отрезке от -4 до 45. Значения функции в промежуточных точках вычислялись линейной интерполяцией/экстраполяцией. В программе заданы толщина линий графика и рамки. Цвет графика выбран красным (цвет номер 5).

В случае *n*-мерной линейной интерполяции используют функцию linear_interpn, имеющую следующий синтаксис:

vp = linear_interpn(xp1,xp2,..,xpn, x1, ..., xn, v [,out_mode])

Подробно о параметрах функции можно узнать в [16] или в справке Scilab. Покажем ее работу в случае функции одной переменной.

Пример 3.6. Функция задана в виде таблицы:

x	0.5	0.55	0.65	0.67	0.98	0.99	1.04	1.34
y	1.2	1.26	1.34	1.34	1.23	1.18	1.01	0.56

С помощью линейной интерполяции вычислить значения функции в точках 0.4, 0.87 и 1.45. Построить график, на который нанести исходные данные и ломаную, полученную линейной интерполяцией. График построить на отрезке [0, 2], разбив его 40 равноотстоящими точками. Принять значение параметра *out_mode* равным "by_zero".

Решение. Один из возможных способов решения данной задачи следующий:

```
clf()
x =[0.5 0.55 0.65 0.67 0.98 0.99 1.04 1.34];
y = [1.2  1.26 1.34 1.34 1.23 1.18 1.01 0.56];
xx = linspace(0,2,40)';yy = linear_interpn(xx, x, y, "by_zero");
plot2d(xx,yy,style=2);plot2d(x,y,style=-9, strf="000")
gca.children.children(1).thickness=3
legend( "Интерполяционная кривая","Точки интерполяции",5)
gca.children.children(2).thickness=3;
gca.thickness=3
xx1=[0.4 0.87 1.45];
yy1=linear_interpn(xx1, x, y, "by_zero");
disp("   x       y")
disp([xx1;   yy1]')
```

Получим:

```
"   x       y"
  0.4    0.
  0.87   1.2690323
  1.45   0.
```

Значения функции за пределами таблицы при таком значении параметра *out_mode* равны нулю.

На рис. 3.7 показан график интерполяционной кривой.

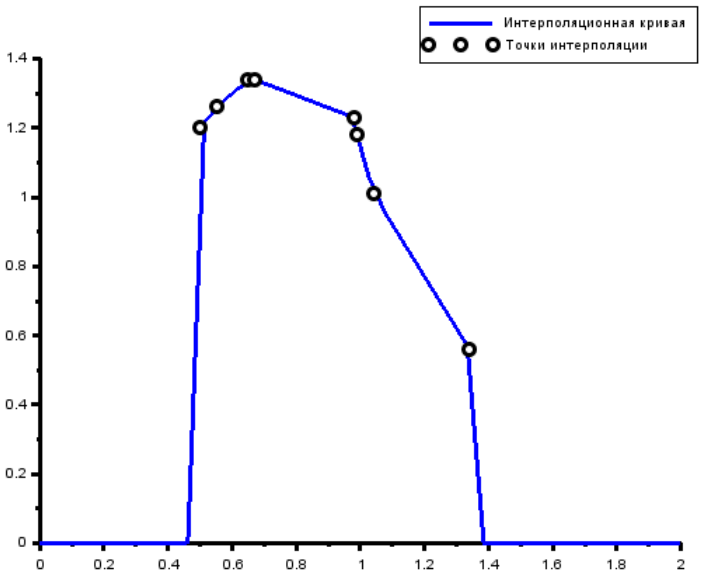


Рис. 3.7. Линейная интерполяция. Значения параметра out_mode=" by_zero"

Примеры с другими значениями параметра *out_mode* приведены в [16].

3.5.1.3. Нахождение коэффициентов интерполяционных полиномов

Покажем, как можно было решить пример 3.2 в Scilab. В приведенной ниже программе значения коэффициентов получаются в результате решения системы из 5 линейных уравнений (полином проходит через 5 точек таблицы).

```
clc;clf()
x =[498 598 698 798 898];
y = [78.03 89.02 98.91 107.76 115.62];
n=length(x);a=[];b=[];c=[];
for i=1:n
  for j=1:n
    a(i,j)=sum(x.^(i+j-2));
  end
  b(i)=sum(x.^(i-1).*y);
end
c=inv(a)*b
function z=f(t)
  z=0;
  for i=1:n
 z=z+t.^(i-1).*c(i);
  end
  endfunction
plot2d(x,y,-4)
t=min(x):0.1:max(x)+.1;plot2d(t,f(t),1);aa=gca();
aa.data_bounds=[min(x)-0.1 77;max(x)+0.1 116];
gca.children.children.thickness=3
r=[870 522];z1=f(r),plot(r,z1,"*");err=sum((y-f(x)).^2)
legend('Исходные данные','Интерполяционный полином', 'Решение задачи интерполяции',5)
 gca.children(1).font_style = 5; gca.children(1).font_size = 3;
 gca.children.children(1).polyline_style = 3;
gca.children.children(1).thickness = 3;
disp('   T        Cp' ;[r;z1]', 'err=' ,err, 'c=' ,c)
```

Строится полином четвертой степени, проходящий через пять заданных точек. Строится график полученной кривой, на который наносятся и исходные данные. Вычисляется значение интерполяционного полинома в заданных точках *r* (переменная *z1*). Ре-

шения задачи интерполяции (значения функции в заданных точках T=522 и T=870) также наносятся на график.

Результаты работы программы:

" T Cp"
870. 113.51649
522. 80.7715
"c="
4.15075
0.1852081
-0.0000833
2.080D-08
-4.167D-12
"err=" 1.840D-12

Коэффициенты полинома равны элементам вектора ***c*** (в порядке возрастания степеней). Правильность решения контролируется переменной ***err***, которая должна принимать значение, близкое к нулю.

На рис.3.8 приведен график получившегося полинома.

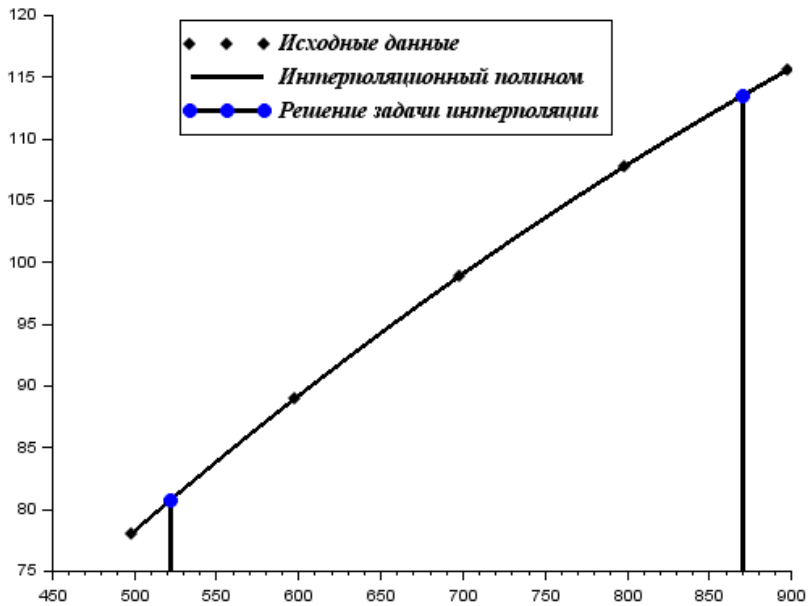


Рис. 3.8. Графическое решение задачи интерполяции по условиям примера 3.2

Замечание. Практика показала, что использование функции *datafit* при решении задачи интерполяции, когда степень аппроксимирующего полинома велика, в некоторых случаях дает неверные результаты.

3.5.1.4. Решение задачи обратной интерполяции

Покажем, как решить пример 3.3 в Scilab.

Первый способ (см. с. 35).

```
clc;clear;clf()
x=[2 4 8 16];y=[1.0146 1.0311 1.0646 1.1342];n=length(x);
for i=1:n
  for j=1:n
    a(i,j)=sum(x.^(i+j-2));
  end
  b(i)=sum(x.^(i-1).*y);
end
c=inv(a)*b
function z=f(t)
  z=0;
  for i=1:n
 z=z+t.^(i-1).*c(i);
  end
  endfunction
plot2d(x,y,-4);t=min(x):0.1:max(x)+.1;plot2d(t,f(t),3)
aa=gca();aa.data_bounds=[min(x)-0.1 1;max(x)+0.1 1.2];
gca.children.children.thickness=3
z1=1.1031 ;
for i=1:n
    zz(i)=c(n-i+1)
end
zz(n)=zz(n)-z1;r=roots(zz);plot(r(3),z1,"*");disp(c)
disp("Решение задачи обратной интерполяции")
disp(r);err=sum((y-f(x)).^2); disp('err=',err)
```

В программе строится интерполяционный полином, проходящий через 4 точки таблицы. Его коэффициенты – вектор **c**:

```
c  =
   0.9982381
   0.00815
   0.0000146
   0.0000004
```

Уравнение полинома

$$y = 0.0000004x^3 + 0.0000146x^2 + 0.00815x + 0.9982381$$

```
"err="   2.024D-23
```

Значение err, близкое к нулю, свидетельствует о том, что полином построен правильно.

Далее решается уравнение

$$0.0000004x^3 + 0.0000146x^2 + 0.00815x + 0.9982381 = 1.1031$$

или

$$0.0000004x^3 + 0.0000146x^2 + 0.00815x + 0.9982381 - 1.1031 = 0$$

Для решения уравнения использовалась функция roots. Поскольку применение этой функции требует, чтобы коэффициенты полинома шли в обратном по отношению к вектору ***c*** порядке, в программе вектор ***c*** был преобразован в вектор zz, из последнего значения которого вычли число 1.1031. Были найдены 3 корня уравнения:

-22.573952 + 135.31415i

-22.573952 - 135.31415i

12.481237 + 0.i

Из трех найденных корней уравнения лишь один – действительный. Он (12.481237) и является решением задачи обратной интерполяции. В программе построен график (рис. 3.9), на который нанесены исходные данные, график самого интерполяционного полинома и решение – точка синего цвета с координатами (12.481237, 1.1031).

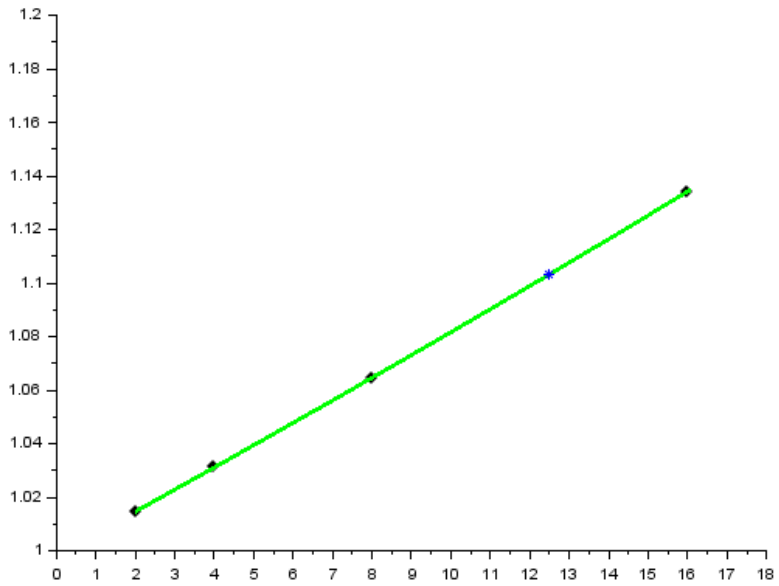


Рис. 3.9. Решение задачи обратной интерполяции первым способом

Второй способ. Строим полином третьей степени. Аргументом является *y*, а функцией – *x*. В получившийся полином подставляем значение z=1.1031.

```
clc;clear;clf();
y=[2 4 8 16];
x=[1.0146 1.0311 1.0646 1.1342];
n=length(x);
a=[];b=[];c=[];
for i=1:n
  for j=1:n
    a(i,j)=sum(x.^(i+j-2));
  end
  b(i)=sum(x.^(i-1).*y);
end
c=inv(a)*b
function z=f(t)
  z=0;
  for i=1:n
 z=z+t.^(i-1).*c(i);
  end
  endfunction
plot2d(x,y,-4)
t=min(x):0.1:max(x)+.1;plot2d(t,f(t),3)
aa=gca();aa.data_bounds=[min(x) 1;max(x) 19];
gca.children.children.thickness=3
z1=1.1031;
disp(f(z1),"Решение задачи обратной интерполяции ")
plot(z1,f(z1),"*")
err=sum((y-f(x)).^2); disp('err=',err)
xlabel("Y");ylabel("X")
Решение:
 c  =
  -92.898621
   4.3619995
   147.95477
  -59.202499
Решение задачи обратной интерполяции
   12.482238
 "err="   7.785D-09
```

Коэффициенты получившегося полинома – это вектор ***c***. Уравнение полинома $x=-92.898621+4.3619995y+147.95477y^2-59.202499y^3$. Его значение x(1.1031)= 12.482238.

Таким образом, ρ=1,1031 при ***C***=12,48%.

В данном примере ответы, полученные первым и вторым способом, совпали, однако так бывает далеко не всегда.

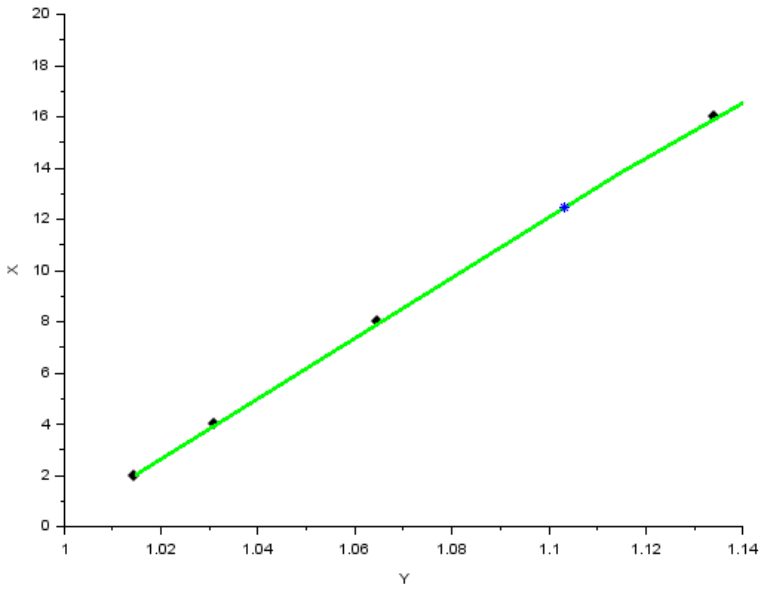


Рис. 3.10. Решение задачи обратной интерполяции вторым способом

3.5.2. Сплайн-интерполяция

3.5.2.1. Построение сплайнов без использования специальных функций Scilab

Покажем, как можно решить пример 3.4 в среде Scilab. Значения искомых коэффициентов в приведенной ниже программе получены в результате решения системы линейных уравнений, приведенной на с. 42. Вместо переменных *a, b, c, d,* a_1, b_1, c_1, d_1 в программе использованы переменные z(1), z(2), … , z(8). z(1) – это *a*, z(2) – это *b*, … , z(8) – это d_1:

```
clc;scf(2);clf();x=[1 2 4];y=[8 12 6];
M=[1 1 1 1 0 0 0 0
  1 2 4 8  0 0 0 0
  0 0 0 0 1 2 4 8
  0 0 0 0 1 4 16 64
  0 1 4 12 0 -1 -4 -12
  0 0 2 12 0 0 -2 -12
```

```
  0 1 2 3 0 0 0 0
  0 0 0 0 0 1 8 48];
  P=[8 12 12 6 0 0 0 0];  z=P/M';  disp(z);  x1=1:.01:2;
  S1=z(1)+z(2)*x1+z(3)*x1.^2+z(4)*x1.^3;
  x2=2:.1:4;S2=z(5)+z(6)*x2+z(7)*x2.^2+z(8)*x2.^3
plot2d(x,y,-4);plot(x1,S1,x2,S2)
gca.data_bounds=[min(x1)-0.1 5;max(x2)+0.1 13];
gca.thickness=3;gca.children.children.thickness=3
```

Получаем значения вектора z:

23. -35.5 26. -5.5 -38. 56. -19.75 2.125

В итоге

$S_1(x)=23-35.5x+26x^2-5.5x^3$

$S_2(x)=-38+56x-19.75x^2+2.125x^3$

3.5.2.2. Построение сплайнов с использованием специальных функций среды Scilab

Построение кубического сплайна с помощью специальных функций в *Scilab* состоит из двух этапов: вначале вычисляются значения первой производной в узлах таблицы с помощью функции *splin*, затем строится кубический сплайн и рассчитываются его значения в заданных точках с помощью функции *interp*.

Функция *d=splin(x,y [,spline_type [, der]])* в зависимости от типа сплайна вычисляет его первые производные в узлах таблицы. Параметры функции: *x* и *y* – исходные данные. *x* - строго возрастающий вектор, состоящий минимум из двух компонент, *y* - вектор той же размерности, что и *x*. Необязательными параметрами являются тип сплайна (*spline_type)* и значение производной (*der* – если выбран тип сплайна 'clamped'). *d* – результат работы функции – значения первой производной в точках *x*.

Тип сплайна может быть "not_a_knot" (по умолчанию), "clamped", "natural", "periodic", "monotone", "fast" и "fast_periodic". Они по-разному задают два недостающих условия в системе уравнений для определения коэффициентов сплайнов (см. с. 41).

Если задан тип сплайна "not_a_knot", то предполагается равенство третьих производных соседних сплайнов во второй и предпоследней точках таблицы; тип сплайна "clamped" предполагает задание конкретных значений первых производных в первой и последней точках таблицы с помощью параметра *der* (например, *splin(x,y, 'clamped',[1,2])*, тип "natural" предполагает равенство нулю

вторых прозводных в первой и последней точках таблицы и т.д. – см. help в Scilab или [16].

Параметры функции *interp*(*t, x, y, d*): *t* – вектор, состоящий из точек, в которых необходимо вычислить значения функции и (при необходимости) ее производных, *d* – это результат работы функции *splin*. С помощью функции *interp* можно вычислять значения первой, второй и третьей производной от сплайна в точках *t*. В этом случае ее надо записать так: *[s0,s1,s2,s3]=interp(t,x,y,d)*. Результат работы таким образом вызываемой функции: *s0* – значения сплайна в точках *t; s1, s2, s3* – значения первой второй и третьей производной в этих же точках.

Примечание. Значение третьей производной от кубического сплайна равно константе. Если вывести на экран значения третьей производной в точках *x*, то первые два значения будут одинаковыми – это значение третьей производной от первого сплайна. Третье значение вектора *s3* – значение третьей производной от второго сплайна, последнее значение вектора *s3* – значение третьей производной от последнего сплайна (см. результаты работы приведенной ниже программы).

Визуально различия сплайнов разного типа можно увидеть на рис. 3.11.

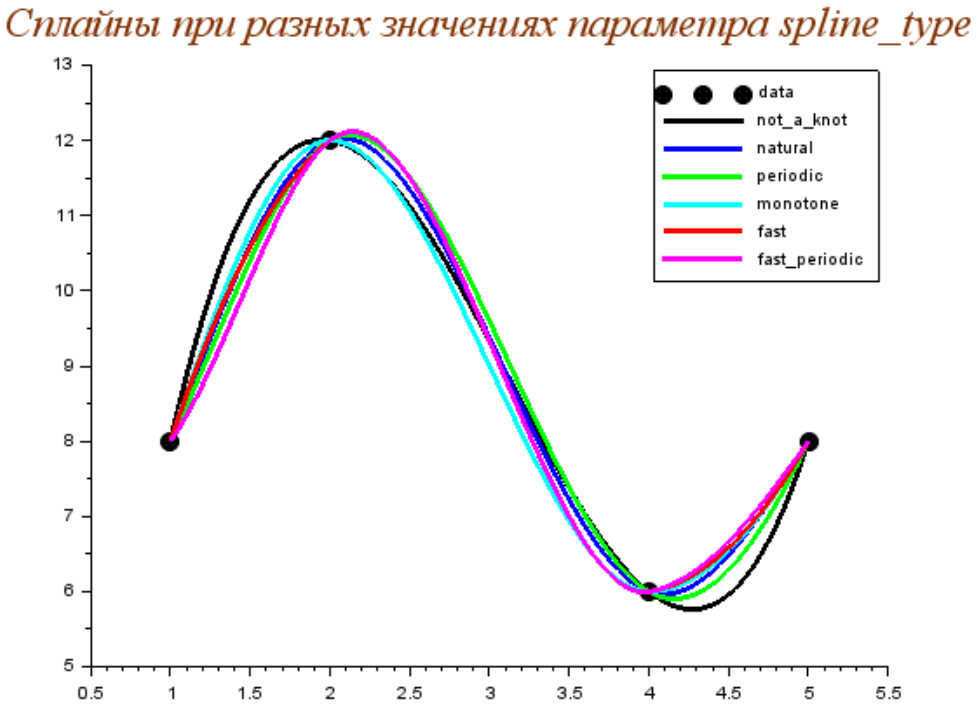


Рис. 3.11. Графики кубических сплайнов разного типа, построенных по одним и тем же точкам таблицы

Покажем, как решить пример 3.4 с помощью функций *splin* и *interp*. Код программы для решения задачи 3.4 следующий:

```
clc;scf(3);clf()
x=[1 2 4];y=[8 12 6];
n=length(x);
deriv=splin(x,y,'clamped',[0,0])//задаем первую производную на концах отрезка
[s0,s1,s2,s3]=interp(x,x,y,deriv);
plot2d(x,y,-3); //Наносим точки на график
//Построение кубических сплайнов на каждом из отрезков
t=min(x):0.01:max(x);
ptd=interp(t,x,y,deriv);
plot2d(t,ptd);
gca.data_bounds=[min(x)-0.1 5;max(x)+0.1 13];
gca.thickness=3
gca.children.children.thickness=3
```

Если вывести на экран значения *s0*, *s1*, *s2*, *s3*, получим:

s0 = 8. 12. 6. – значения сплайна в узлах таблицы;

s1 = 0. 2.5 0. – первые производные в узлах таблицы;

s2 = 19. -14. 11.5 – вторые производные в узлах таблицы;

s3 = -33. -33. 12.75 – третьи производные (константы, равные 6d (или 6d1): -5.5*6=-33 (6*2.125=12.75)).

Графики полученных сплайнов были приведены на рис. 3.5.

Коэффициенты сплайнов в этой программе не вычисляются. Покажем, как в можно найти коэффициенты сплайна, если он представлен в виде (3.8), воспользовавшись тем, что функция *interp* может вычислять производные сплайна в заданных точках. Выберем тип сплайна “natural”.

Напоминаем, что

$$S_i'(x) = b_i + 2c_i(x - x_{i-1}) + 3d_i(x - x_{i-1})^2$$

$$S_i''(x) = 2c_i + 6d_i(x - x_{i-1}),\ S_i'''(x) = 6d_i$$

$$S_i'(x_{i-1}) = b_i,\ S_i'(x_i) = b_i + 2c_ih_i + 3d_ih_i^2,\ S_i''(x_{i-1}) = 2c_i,\ S_i''(x_i) = 2c_i + 6d_ih_i,$$

$$h_i = x_i - x_{i-1}, \quad i = \overline{1, n}.$$

То есть, зная значения первой, второй и третьей производной, нетрудно определить значения искомых коэффициентов сплайна. Если нам известны значения только первых производных, то коэффициенты a_i, b_i, c_i и d_i в формуле (3.8) можно вычислить так:

$$a_i = S_i(x_{i-1})$$

$$b_i = S_i'(x_{i-1})$$

$$c_i = \frac{3S_i(x_i) - 3S_i(x_{i-1}) - 2S_i'(x_{i-1})h_i - S_i'(x_i)h_i}{h_i^2}$$

$$d_i = \frac{S_i'(x_i) - S_i'(x_{i-1}) - 2c_i h_i}{3h_i^2}.$$

Здесь $h_i = x_i - x_{i-1}$.

Исходя из этого, можно найти значения коэффициентов:

```
clc;clf()
x=[1 2 4];
y=[8 12 6];
n=length(x);
deriv=splin(x,y,'natural');//Вычисление значений функции и
//производных в заданных точках
[s0,s1,s2,s3]=interp(x,x,y,deriv);
plot2d(x,y,-3); //Нанесение точек на график
//Построение кубических сплайнов на каждом из отрезков
t=min(x):0.01:max(x);ptd=interp(t,x,y,deriv);plot2d(t,ptd);
//Вычисление коэффициентов полиномов
for p=1:n-1
  h(p)=x(p+1)-x(p);
  a(p+1)=s0(p);
  b(p+1)=s1(p);
c(p+1)=(3*s0(p+1)-3*s0(p)-2*s1(p)*h(p)-
s1(p+1)*h(p))/(h(p)^2);
   d(p+1)=(s1(p+1)-s1(p)-2*c(p+1)*h(p))/(3*h(p)^2);
end
for p=1:n-1;
  printf("Коэффициенты полинома  ");printf('%2.0f  ',p);
printf('%3.4f %3.4f %3.4f %3.4f \n',
a(p+1),b(p+1),c(p+1),d(p+1))
end
//Построение графиков получившихся сплайнов
for i=1:n-1
z=x(i):0.01:x(i+1);
  r=a(i+1)+b(i+1)*(z-x(i))+c(i+1)*(z-x(i))^2+d(i+1)*(z-x(i))^3;
  plot(z,r,'r')
```

```
    end
aa=gca();
aa.data_bounds=[min(x)-0.01 5;max(x)+0.01 14];
gca.children.children.thickness=3
```

Результаты работы программы:

Коэффициенты полинома 1 8.0000 5.1667 0.0000 -1.1667

Коэффициенты полинома 2 12.0000 1.6667 -3.5000 0.5833

Таким образом, на участке [1, 2] построен сплайн

$$S_1(x) = 8 + 5.1667\ (x-1) - 1.1667(x-1)^3,$$

а на отрезке [2 , 4]

$$S_2(x) = 12 + 1.6667(x-2) - 3.5\ (x-2)^2 + 0.5833(x-2)^3.$$

Здесь $h_i = x_i - x_{i-1}$.

Заметим, что уравнения сплайнов, приведенных выше, получились отличными от тех, что приведены на с. 53. Различие можно увидеть, сравнив графики (рис. 3.12) получившихся там и здесь сплайнов.

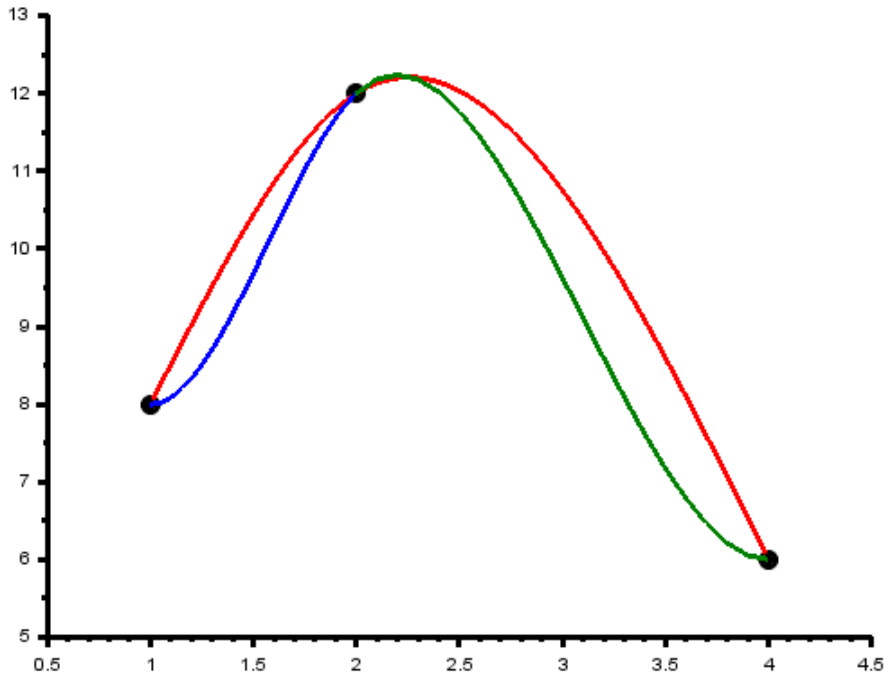


Рис. 3.12. Графики кубических сплайнов решения примера 3.4 при разных вариантах задания значений производных

Это связано с тем, что при решении примера 3.4 по-разному задавались два последних условия в системе уравнений, приведенной на с. 41. В первом случае полагалось, что

$$S_1^{'}(x_0)=0,\ S_2^{'}(x_2)=0.$$

Во втором - при выборе типа сплайна *'natural'* предполагалось, что

$$S_1^{''}(x_0)=0,\ S_2^{''}(x_2)=0.$$

Для того, чтобы определить коэффициенты сплайна, представленного в виде $S(x)=a+bx+cx^2+dx^3$, можно модифицировать приведенную выше программу, как это выполнено в [16].

В заключение покажем, как можно аппроксимировать сплайнами явно заданную функцию, не используя функцию *splin*. Пусть, например, это функция, рассмотренная в конце раздела 2.3: *y=sin(x)+sin(2x).* Тогда задача интерполяции решалась с помощью интерполяционных полиномов. Отрезок, на котором будем проводить аппроксимацию – [1, 10]. Для более удобного визуального анализа различий при применении двух способов интерполяции – с помощью интерполяционных полиномов и сплайнов – объединим два графика.

```
clear;clc
x=1:10;
function y=f(x)
y=sin(x)+sin(2*x);
endfunction
//Создание вектора значений первой производной в
//точках x=1,2,3,...,10
scf(2);clf()
n=length(x);a=[];b=[];c1=[];
for i=1:n
  for j=1:n
    a(i,j)=sum(x.^(i+j-2));
  end
  b(i)=sum(x.^(i-1).*f(x));
end
c1=inv(a)*b
function z=f1(t)
  z=0;
  for i=1:n
 z=z+t.^(i-1).*c1(i);
  end
  endfunction
err=sum((f(x)-f1(x)).^2)
```

```
        for i=1:n
t(i)=numderivative(f,x(i));//Вычисление производной от заданной
//функции y=sin(x)+sin(2*x);
         end
         //Вычисление коэффициентов сплайнов
        for p=1:n-1
         h(p)=x(p+1)-x(p);
         a(p+1)=f(p);
         b(p+1)=t(p);
        c(p+1)=(3*f(p+1)-3*f(p)-2*t(p)*h(p)-t(p+1)*h(p))/(h(p)^2);
           d(p+1)=(t(p+1)-t(p)-2*c(p+1)*h(p))/(3*h(p)^2);
        end
        disp('Вектор коэффициентов сплайна в порядке возрастания
степеней ')
        for p=1:n-1;
           printf('Коэффициенты сплайна  ');printf('%2.0f  ',p);
printf('  %3.4f  %3.4f  %3.4f  %3.4f\n ', a(p+1),b(p+1),c(p+1),d(p+1))
            disp('')
        end
        plot2d(x,f(x),-3); //Наносим точки на график
        //Построение графиков получившихся сплайнов, заданной
функции и //интерполяционного полинома
        for i=1:n-1
        z1=x(i):0.01:x(i+1);
         r1=a(i+1)+b(i+1)*(z1-x(i))+c(i+1)*(z1-x(i))^2+d(i+1)*(z1-
x(i))^3;
          plot(z1,r1,'r', z1, f(z1),'--b',z1,f1(z1),'g')
          gca.children.children.thickness=3
         end
         err=sum((f(x)-f1(x)).^2)
        legend('data', 'Сплайн', 'y=sin(x)+sin(2x)','Полином')
        aa=gca();aa.data_bounds=[0.5 -2;10.5 3.5];
```

Результаты работы программы.

"Вектор коэффициентов сплайна в порядке возрастания степеней "

Коэффициенты сплайна	1	1.7508	-0.2920	-2.4874	1.1811
Коэффициенты сплайна	2	0.1525	-1.7234	1.6441	-0.2115
Коэффициенты сплайна	3	-0.1383	0.9303	0.1965	-0.7560
Коэффициенты сплайна	4	0.2326	-0.9446	-1.9227	1.1319
Коэффициенты сплайна	5	-1.5029	-1.3945	2.2020	-0.1205

Коэффициенты сплайна 6 -0.8160 2.6479 1.0676 -1.2519
Коэффициенты сплайна 7 1.6476 1.0274 -2.8324 0.8588
Коэффициенты сплайна 8 0.7015 -2.0608 0.5912 0.4293
Коэффициенты сплайна 9 -0.3389 0.4095 1.3273 -1.0290

Так, уравнение первого сплайна

$$S_1(x) = 1.7508 - 0.2920\,(x-1) - 2.4874\,(x-1)^2 + 1.1811(x-1)^3,$$

уравнение второго сплайна

$$S_2(x) = 0.1525 - 1.7234\,(x-2) + 1.6441\,(x-2)^2 - 0.2115(x-2)^3,$$

уравнение последнего

$$S_9(x) = -0.3389 + 0.4095(x-9) + 1.3273\,(x-9)^2 - 1.0290(x-9)^3.$$

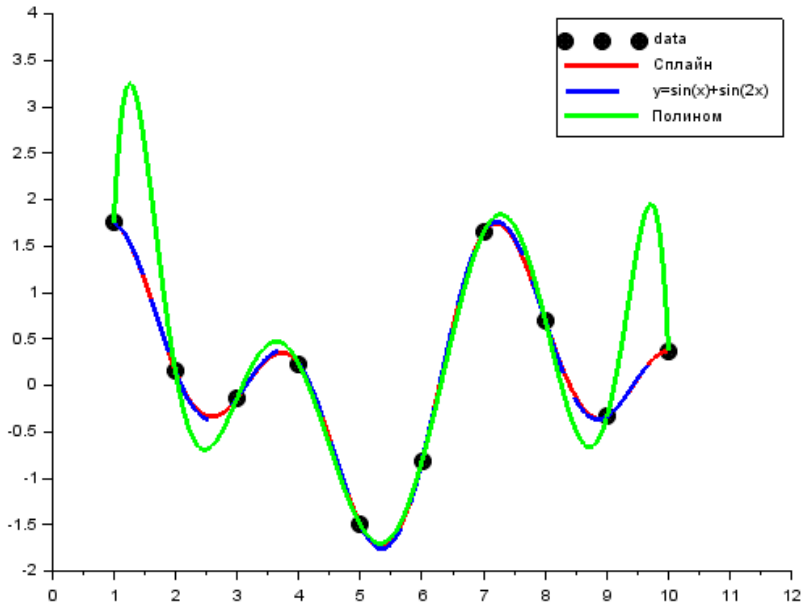


Рис. 3.13. Графики исходной функции, сплайна и интерполяционного полинома

Из рис. 3.13 видно, что аппроксимация (если говорить об отрезке [1, 10]) сплайнами (красная кривая) существенно лучше приближает исходную функцию (синяя кривая), чем интерполяционный полином (зеленая кривая).

4. ЧИСЛЕННОЕ ДИФФЕРЕНЦИРОВАНИЕ

Численное дифференцирование – это вычисление производных от функций, заданных в виде таблицы. Задача решается в два этапа. На первом этапе строится интерполяционный полином. На втором этапе находятся производные (первая, вторая и т.д.) от этого полинома.

4.1. Численное дифференцирование с помощью интерполяционных полиномов

Если шаг в таблице постоянный, то на первом этапе строят интерполяционный полином Ньютона.

$$Q_n^I(x) = y_0 + q\Delta y_0 + \frac{q(q-1)}{2!}\Delta^2 y_0 + \frac{q(q-1)(q-2)}{3!}\Delta^3 y_0 + \ldots$$

$$+\frac{q(q-1)(q-2)\ldots(q-n+1)}{n!}\Delta^n y_0,\ h = x_{i+1} - x_i = const,\ q = \frac{x - x_0}{h}.$$

Найдем первую производную от этого полинома. $\frac{\partial Q}{\partial x} = \frac{\partial Q}{\partial q}\frac{\partial q}{\partial x}$.

$$\frac{\partial Q}{\partial x} = \frac{1}{h}(\Delta y_0 + \frac{2q-1}{2}\Delta^2 y_0 + \frac{3q^2 - 6q + 2}{6}\Delta^3 y_0 + \frac{2q^3 - 9q^2 + 11q - 3}{12}\Delta^4 y_0 +$$

$$+\frac{5q^4 - 40q^3 + 105q^2 - 100q + 24}{120}\Delta^5 y_0 +$$

$$+\frac{6q^5 - 75q^4 + 340q^3 - 675q^2 + 548q - 120}{720}\Delta^6 y_0 + \ldots). \quad (4.1)$$

Аналогично,

$$\{Q_n\}'' = \frac{1}{h^2}[\Delta^2 y_0 + (q-1)\Delta^3 y_0 + \frac{6q^2 - 18q + 11}{12}\Delta^4 y_0 +$$

$$+\frac{2q^3 - 12q^2 + 21q - 10}{12}\Delta^5 y_0 + \frac{30q^4 - 300q^3 + 1020q^2 - 1350q + 548}{720}\Delta^6 y_0 + \ldots]. \quad (4.2)$$

Если производную нужно вычислить в каком-либо узле таблицы (например, в точке $x=x_0$), то формулы (4.1) и (4.2) упрощаются, так как $q_1 = \frac{x - x_0}{h} = 0$.

$$\frac{\partial Q}{\partial x}|_{x=x_0} = \frac{1}{h}\left(\Delta y_0 - \frac{1}{2}\Delta^2 y_0 + \frac{1}{3}\Delta^3 y_0 - \frac{1}{4}\Delta^4 y_0 + \frac{1}{5}\Delta^5 y_0 - \frac{1}{6}\Delta^6 y_0 + \ldots\right) \quad (4.3)$$

$$\{Q_n\}''|_{x=x_0} = \frac{1}{h^2}[\Delta^2 y_0 - \Delta^3 y_0 + \frac{11}{12}\Delta^4 y_0 - \frac{10}{12}\Delta^5 y_0 + \frac{137}{180}\Delta^6 y_0 + \ldots]. \quad (4.4)$$

Пример 4.1. Зависимость вязкости (η) нитробензола от температуры выражается следующими экспериментальными данными [5]:

T, K	283,15	293,15	303,15	313,15	323,15	333,15	343,15
η, мПа·с	2,509	2,013	1,682	1,438	1,251	1,094	0,970

Найти абсолютный температурный коэффициент вязкости $\frac{\partial \eta}{\partial T}$ а) при T=288,15 К; b) при температуре T=283,15 К.

Решение. Так как шаг в приведенной выше таблице – постоянный, для решения задачи воспользуемся формулой (4.1). Составим таблицу конечных разностей:

T	η	$\Delta\eta$	$\Delta^2\eta$	$\Delta^3\eta$	$\Delta^4\eta$	$\Delta^5\eta$	$\Delta^6\eta$
283,15	2,509	-0,496	0,165	-0,078	0,048	-0,045	0,072
293,15	2,013	-0,331	0,087	-0,03	0,003	0,027	
303,15	1,682	-0,244	0,057	-0,027	0,03		
313,15	1,438	-0,187	0,03	0,003			
323,15	1,251	-0,157	0,033				
333,15	1,094	-0,124					
343,15	0,97						

Имеем:

$$\frac{\partial \eta}{\partial T} = \frac{1}{h}(\Delta\eta_0 + \frac{2q-1}{2}\Delta^2\eta_0 + \frac{3q^2-6q+2}{6}\Delta^3\eta_0 + \frac{2q^3-9q^2+11q-3}{12}\Delta^4\eta_0 +$$

$$+\frac{5q^4-40q^3+105q^2-100q+24}{120}\Delta^5\eta_0 + \frac{6q^5-75q^4+340q^3-675q^2+548q-120}{720}\Delta^6\eta_0).$$

$$h=10,\ q=\frac{T-T_0}{h}=\frac{288{,}15-283{,}15}{10}=0{,}5.$$

Подставляя в приведенную формулу данные из таблицы, получаем:

$$\frac{\partial \eta}{\partial T}|_{T=288.15} = \frac{1}{10}[-0{,}496 + \frac{2\cdot 0{,}5-1}{2}\cdot 0{,}165 + \frac{3\cdot 0{,}5^2-6\cdot 0{,}5+2}{6}\cdot(-0{,}078) +$$

$$+\frac{2\cdot 0{,}5^3-9\cdot 0{,}5^2+11\cdot 0{,}5-3}{12}\cdot(0{,}048) +$$

$$+\frac{5\cdot 0,5^4-40\cdot 0,5^3+105\cdot 0,5^2-100\cdot 0,5+24}{120}(-0,045)+$$

$$+\frac{6\cdot 0,5^5-75\cdot 0,5^4+340\cdot 0,5^3-675\cdot 0,5^2+548\cdot 0,5-120}{720}(0,072)]=-0,0486761.$$

Найдем теперь значение производной в узле таблицы – при T=283,15.

В этом случае $q=\frac{T-T_0}{h}=\frac{283,15-283,15}{10}=0$ и можно воспользоваться формулой (4.3).

$$\frac{\partial\eta}{\partial T}=\frac{1}{h}(\Delta\eta_0-\frac{1}{2}\Delta^2\eta_0+\frac{1}{3}\Delta^3\eta_0-\frac{1}{4}\Delta^4\eta_0+\frac{1}{5}\Delta^5\eta_0-\frac{1}{6}\Delta^6\eta_0.$$

Имеем:

$$\frac{\partial\eta}{\partial T}|_{T=283.15}=\frac{1}{10}[(-0,496)-\frac{1}{2}(0.165)+\frac{1}{3}(-0,078)-\frac{1}{4}(0,048)+\frac{1}{5}(-0,045)-$$

$$-\frac{1}{6}(0,072)]=-0,06375.$$

Данный пример приведен исключительно в иллюстративных целях, чтобы показать, как работают формулы численного дифференцирования. При решении реальной задачи, связанной с вычислением производных, лучше, на наш взгляд, получить уравнение интерполяционного полинома, как это сделано в разделе 3, а затем вычислить от него производную/ые. Можно воспользоваться также программой, приведенной на с. 66.

Для вычисления значений производных в узлах таблицы в зависимости от количества узлов и расположения точки, в которой нужно вычислить производную, существуют формулы численного дифференцирования, не такие громоздкие, как формулы (4.1) и (4.2). Приведем некоторые из них (см. табл. 4.1). Шаг в таблице должен быть постоянным.

Отметим, что численное дифференцирование относится к числу некорректных задач. Это означает, что небольшие погрешности в исходных данных могут привести к большим погрешностям в результате решения задачи. По этой же причине следует крайне осторожно пользоваться формулами численного дифференцирования при вычислении производных старших порядков.

Таблица 4.1

Формулы численного дифференцирования

Формула $y_i = y(x_0 + ih),\ y_i' = y'(x_0 + ih)$	Оценка погрешности
$y_0' = \frac{1}{2h}(-y_{-1} + y_1)$	$-\frac{1}{6}h^2 y_0''' - \ldots$
$y_0' = \frac{1}{12h}(y_{-2} - 8y_{-1} + 8y_1 - y_2)$	$+\frac{1}{30}h^4 y_o^{(5)} + \ldots$
$y_0' = \frac{1}{h}(-y_0 + y_1)$	$-\frac{1}{2}hy_0'' - \ldots$
$y_0' = \frac{1}{2h}(-3y_0 + 4y_1 - y_2)$	$\frac{1}{3}h^2 y_0''' + \ldots$
$y_0' = \frac{1}{12h}(-3y_{-1} - 10y_0 + 18y_1 - 6y_2 + y_3)$	$-\frac{1}{20}h^4 y_o^{(5)} + \ldots$
$y_0'' = \frac{1}{h^2}(y_{-1} - 2y_0 + y_1)$	$-\frac{1}{12}h^2 y_o^{(4)} + \ldots$
$y_0'' = \frac{1}{12h^2}(-y_{-2} + 16y_{-1} - 30y_0 + 16y_1 - y_2)$	$\frac{1}{90}h^4 y_o^{(6)} + \ldots$
$y_0'' = \frac{1}{h^2}(2y_0 - 5y_1 + 4y_2 - y_3)$	$\frac{11}{12}h^2 y_o^{(4)} + \ldots$
$y_0'' = \frac{1}{12h^2}(11y_{-1} - 20y_0 + 6y_1 + 4y_2 - y_3)$	$\frac{1}{12}h^3 y_o^{(5)} + \ldots$
$y_0'' = \frac{1}{180h^2}(-13y_{-2} + 228y_{-1} - 420y_0 + 200y_1 + +15y_2 - 12y_3 + 2y_4)$	$-\frac{1}{90}h^5 y_o^{(7)} + \ldots$
$y_0''' = \frac{1}{2h^3}(-y_{-2} + 2y_{-1} - 2y_1 + y_2)$	$-\frac{1}{4}h^2 y_0^{(5)} + \ldots$
$y_0''' = \frac{1}{2h^3}(-3y_{-1} + 10y_0 - 12y_1 + 6y_2 - y_3)$	$+\frac{1}{4}h^2 y_0^{(5)} + \ldots$

Пример 4.2. Функция задана следующей таблицей.

x	1	2	3	4	5
y	1.7475649	4.1428558	7.4540643	13.888067	28.290145

Вычислить значение производной в точке x=2 по одной из приведенных в табл. 4.1 формул.

Решение. Выберем формулу, содержащую 5 точек таблицы.

$$y_0' = \frac{1}{12h}(-3y_{-1} - 10y_0 + 18y_1 - 6y_2 + y_3) =$$

$$= \frac{1}{12 \cdot 1}(-3 \cdot 1.7475649 - 10 \cdot 4.1428558 + 18 \cdot 7.4540643 - 6 \cdot 13.888067 +$$

$$+28.290145) = 2.7053041.$$

Таблица в примере 4.2 была получена табулированием функции $y = xlnx + sin(x) + \frac{e^x}{x+2}$. Производная от этой функции равна $y' = lnx + 1 + \cos(x) + \frac{e^x(x+1)}{(x+2)^2}$. Точное значение производной равно 2.6624484. Относительная погрешность вычислений составляет не более 2 %.

4.2. Вычисление производных в среде Scilab

Пример 4.1. можно решить в среде Scilab с помощью следующей программы:

```
clc
funcprot(0)
clear
T=[283.15 293.15 303.15 313.15 323.15 333.15 343.15];
y=[2.509 2.013 1.682 1.438 1.251 1.094 0.970];
n=length(T);a=[];b=[];c=[];
h=10;
x=(T-T(1))/h;
for i=1:n
  for j=1:n
    a(i,j)=sum(x.^(i+j-2));
  end
  b(i)=sum(x.^(i-1).*y);
end
c=inv(a)*b
function z=f(t)
  z=0;
  for i=2:n
 z=z+(i-1)*((t-T(1))/h).^(i-2).*c(i)/h;
```

```
  end
endfunction
xx=[288.15 283.15];
w=f(xx)
disp([xx;w])
```

Результаты:

```
   288.15        283.15
 -0.0486761  -0.06375
```

Краткий комментарий к программе. Для уменьшения вычислительной погрешности вместо переменной T введена переменная $x = \frac{T - 283.15}{10}$. Попытка решить задачу без такой замены привела к почти 12 % относительной погрешности по отношению к ответу, полученному на с. 66. В программе проводится расчет коэффициентов c_1, c_2, …, c_7 интерполяционного полинома представленного в виде

$$Q_6(T) = c_1 + c_2\left(\frac{T-283.15}{10}\right) + ... + c_7\left(\frac{T-283.15}{10}\right)^6 = \sum_{i=1}^{7} c_i\left(\frac{T-283.15}{10}\right)^{i-1}.$$

От него вычисляется производная, которая равна

$$\frac{c_2}{10} + ... + 6\frac{c_7}{10}\left(\frac{T-283.15}{10}\right)^5 = \frac{1}{10}\sum_{i=2}^{7} c_i(i-1)\left(\frac{T-283.15}{10}\right)^{i-2}.$$

В эту формулу подставлялись точки, в которых нужно было вычислить производную. При необходимости можно вывести на экран коэффициенты полинома – значения вектора ***c***. Если функция задана в явном виде, то производную от нее можно вычислить с помощью функции *numderivative*.

Пример 4.3. Вычислить первую производную от функции $y = xlnx + sin(x) + \frac{e^x}{x+2}$ в точках 1.2, 4.2 и 5.

```
clear
function y=f(x)
   y=x.*log(x)+sin(x)+exp(x)./(x+2);
endfunction
function y1=f1(x)
   y1=log(x)+1+cos(x)+(exp(x).*(x+1))./((x+2).^2);
endfunction
```

```
w=[1.2 4.2 5];v=numderivative(f,w)
disp([w;v;f1(w)])
```

Результаты работы программы:

```
1.2          4.2          5.
2.2579857    0.           0.
0.           10.965867    0.
0.           0.           21.06614
2.2579857    10.965867    21.06614
```

В приведенной выше программе вычисляются значения первой производной в указанных точках (ненулевые значения во второй, третьей и четвертой строках). Последняя строка – точное значение производной в этих же точках.

Решим эту же задачу с помощью интерполяционного полинома. Протабулируем функцию $y = xlnx + sin(x) + \frac{e^x}{x+2}$ на отрезке [1, 6] с шагом 1 и воспользуемся программой, приведенной на с. 66, немного ее модифицировав:

```
clear
function y=f(x)
   y=x.*log(x)+sin(x)+exp(x)./(x+2);
endfunction
w=[1.2 4.2 5];x=[1:6];y=f(1:6);
n=length(y);
for i=1:n
 for j=1:n
   a(i,j)=sum(x.^(i+j-2));
 end
 b(i)=sum(x.^(i-1).*y);
end
c=inv(a)*b
function z=f2(t)
 z=0;
 for i=2:n
z=z+(i-1)*t.^(i-2).*c(i);
 end
endfunction
```

w1=f2(w)
disp([w;w1])
Результат:

1.2	4.2	5.
2.3885201	10.900996	21.211963

Сравнивая полученные значения с точным ответом, видим, что относительная погрешность составляет от 1 до 6 %.

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

Задание 1. Функция задана в виде таблицы.

x	μ+ν	μ+ν+1	μ+ν+4	μ+ν+5	μ+ν+8
y	γ+θ+2.5	γ+θ+4.9	γ+θ+8	γ+θ+12.1	γ+θ+16.9

Используя все точки таблицы, построить интерполяционный полином и вычислить первую и вторую производные от этого полинома в точках $x_1^* = \mu + \nu$ и $x_2^* = \mu + \nu + 1 + Nst / 30$. *Nst* – номер студента по списку.

Задание 2. Функция задана в виде таблицы.

x	μ+ν	μ+ν+1	μ+ν+2	μ+ν+3	μ+ν+4
y	γ+1.3	γ+θ	γ+θ+3	γ+θ+7	γ+θ+10.9

Вычислить первую и вторую производные от этой функции в точке $x_1^* = \mu + \nu + 1$ с помощью формул численного дифференцирования из таблицы 3.1. Использовать все пять точек таблицы.

Задание 3. Функция задана в виде таблицы.

x	μ+ν	μ+ν+2	μ+ν+4	μ+ν+6	μ+ν+8
y	γ+θ+2.5	γ+θ+4.9	γ+θ+8	γ+θ+12.1	γ+θ+16.9

Используя все точки таблицы, вычислить первую и вторую производные от заданной функции по формулам численного дифференцирования в точках $x_1^* = \mu + \nu$ и $x_2^* = \mu + \nu + 1 + Nst / 30$. *Nst* – номер студента по списку.

5. ЧИСЛЕННОЕ ИНТЕГРИРОВАНИЕ

Численное интегрирование (историческое название: *(численная квадратура)*) – это вычисление определенных интегралов от функций, заданных либо в явном виде (например, $\int_0^1 e^{-\frac{x^2}{2}} dx$), либо в виде таблицы. К численному интегрированию прибегают тогда, когда точными методами интеграл вычислить либо невозможно, либо сложно. Например, аналитическое представление подынтегральной функции известно $\left(f(x) = e^{-\frac{x^2}{2}} \right)$, но её первообразная не выражается через аналитические функции. Если подынтегральная функция задана аналитически, то ее приводят к табличному виду, задавая шаг h и вычисляя значения функции в определенных точках. Интегралы вычисляют с помощью так называемых квадратурных формул, то есть

$$\int_a^b f(x)dx \approx \sum_{i=0}^{n} A_i f(x_i) \qquad (5.1)$$

Для нахождения коэффициентов A_i в формуле (5.1) функцию $y = f(x)$ на каждом из отрезков $[x_i, x_{i+1}]$ заменяют интерполяционным полиномом. Интеграл от полинома легко вычисляется. Для нахождения двойных определенных интегралов существуют соответствующие кубатурные формулы (с двойными суммами).

5.1. Квадратурные формулы прямоугольников, трапеций, Симпсона для вычисления определенных интегралов

5.1.1.Формулы прямоугольников, трапеций и Симпсона

Существует несколько формул прямоугольников: формула левых прямоугольников, формулы правых и центральных прямоугольников.

В формуле левых прямоугольников на интервале интегрирования – отрезке $[a, b]$ – подынтегральную функцию $y = f(x)$ заменяют прямой $y = f(a)$ и считают, что интеграл (то есть площадь криволинейной трапеции, ограниченной осью X, прямыми $x=a$, $x=b$ и графиком функции $y = f(x)$) приближенно равен площади прямоугольника, основанием которого является отрезок $[a, b]$, а высотой $f(a)$ – значение подынтегральной функции в точке a:

$$I_{лп} = f(a)(b-a) \qquad (5.2)$$

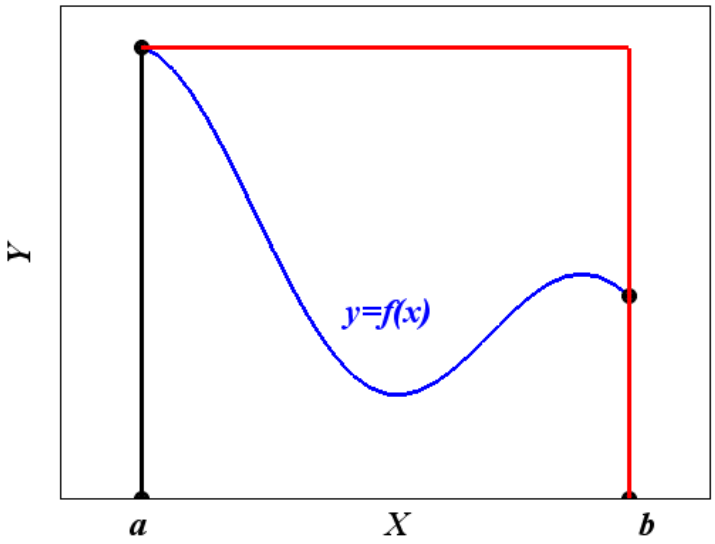


Рис. 5.1. Геометрическая интерпретация формулы левых прямоугольников

В формуле правых прямоугольников на отрезке [*a*, *b*] функцию *y=f(x)* заменяют прямой *y=f(b)* и считают, что интеграл приближенно равен площади прямоугольника, основанием которого является отрезок [*a*, *b*], а высотой *f(b)* – значение подынтегральной функции в точке *b*:

$$I_{пп} = f(b)(b-a) \qquad (5.3)$$

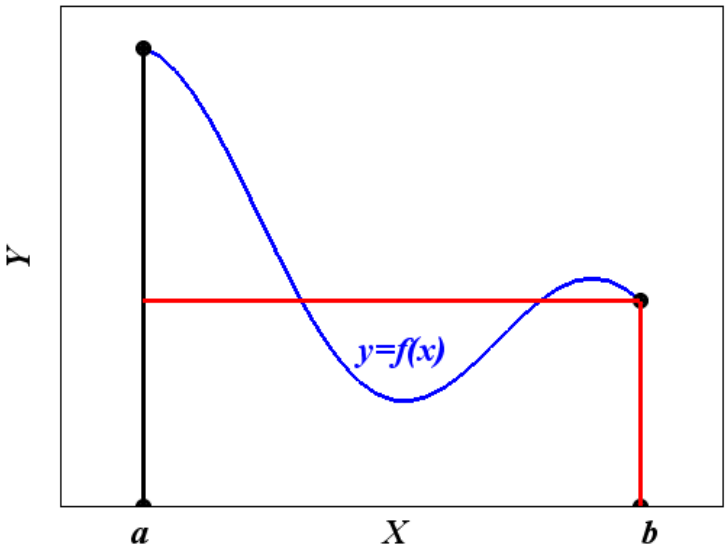


Рис. 5.2. Геометрическая интерпретация формулы правых прямоугольников

В формуле центральных прямоугольников на отрезке [a, b] функцию $y = f(x)$ заменяют прямой $y = f\left(\frac{a+b}{2}\right)$ и считают, что интеграл приближенно равен площади прямоугольника, основанием которого является отрезок [a, b], а высотой значение функции в точке $\left(\frac{a+b}{2}\right)$:

$$I_{цпр} = f\left(\frac{a+b}{2}\right)\cdot(b-a) \qquad (5.4)$$

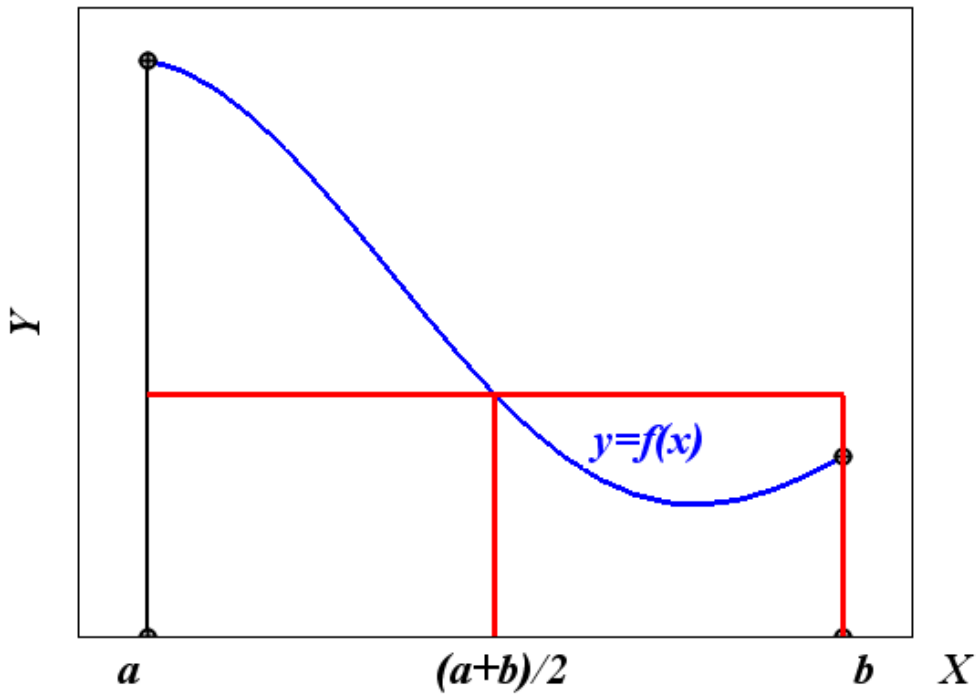


Рис. 5.3. Геометрическая интерпретация формулы центральных прямоугольников

В формуле трапеций на отрезке [a, b] функцию $y=f(x)$ заменяют полиномом первой степени – прямой, проходящей через точки с координатами (a, $f(a)$) и (b, $f(b)$) – и считают, что интеграл приближенно равен площади прямоугольной трапеции, образованной прямыми $x=a$, $x=b$, осью X и прямой, проходящей через точки (a, $f(a)$) и (b, $f(b)$):

$$I_{тр} = \frac{f(a)+f(b)}{2}(b-a) \qquad (5.5)$$

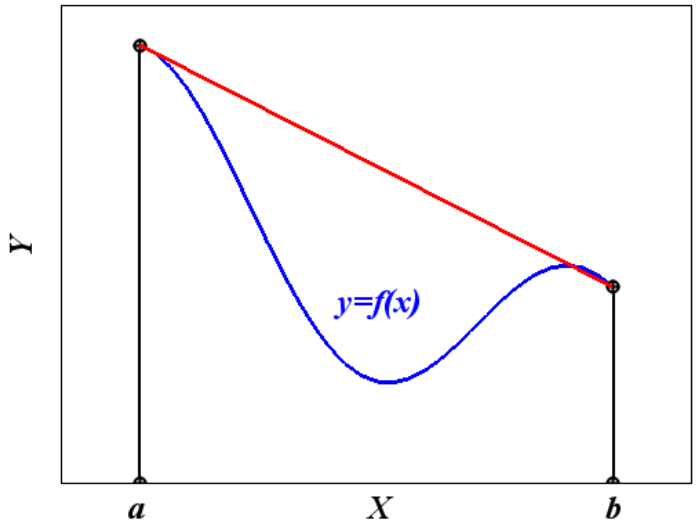


Рис. 5.4. Геометрическая интерпретация формулы трапеций

В формуле Симпсона (формуле парабол) на отрезке $[a, b]$ функцию $y=f(x)$ заменяют полиномом второй степени, проходящим через точки $(a, f(a))$, $\left(\frac{a+b}{2}, f\left(\frac{a+b}{2}\right)\right)$ и $(b, f(b))$ и считают, что интеграл приближенно равен площади криволинейной трапеции, ограниченной осью X, прямыми $x=a$, $x=b$ и параболой, проходящей через указанные точки:

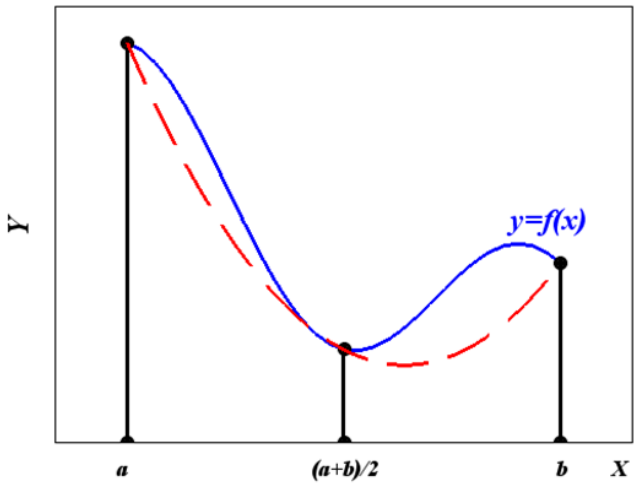


Рис. 5.5. Геометрическая интерпретация формулы Симпсона

Покажем, как выводится формула Симпсона. Запишем уравнение параболы, проходящей через три точки (*a*, *f(a)*) , $\left(\frac{a+b}{2},\ f\left(\frac{a+b}{2}\right)\right)$ и (*b*, *f(b)*), используя формулу интерполяционного полинома Ньютона второй степени:

$$Q_2(x) = y_0 + q\Delta y_0 + \frac{q(q-1)}{2!}\Delta^2 y_0.$$

Здесь

$$q = \frac{x-a}{h},\ h = x_{i+1} - x_i = \frac{b-a}{2},\ y_0 = f(a),\ \Delta y_0 = f\left(\frac{a+b}{2}\right) - f(a),$$

$$\Delta^2 y_0 = f(b) - 2f\left(\frac{a+b}{2}\right) + f(a).$$

Площадь искомой криволинейной трапеции *S* равна

$$S = \int_a^b Q_2(x)dx = \int_0^2 \left(y_0 + q\Delta y_0 + \frac{q(q-1)}{2!}\Delta^2 y_0 \right) h dq.$$

Здесь мы сделали замену переменной интегрирования.
Отсюда

$$S = h\left\{ y_0 q + \frac{q^2}{2}\Delta y_0 + \left(\frac{q^3}{6} - \frac{q^2}{4}\right)\Delta^2 y_0 \right\}_0^2 = h\left(2y_0 + 2\Delta y_0 + \frac{\Delta^2 y_0}{3} \right) =$$

$$= h\left[2f(a) + 2f\left(\frac{a+b}{2}\right) - 2f(a) + \frac{f(a) - 2f\left(\frac{a+b}{2}\right) + f(b)}{3} \right] =$$

$$= \frac{h}{3}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b) \right] = \frac{b-a}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b) \right].$$

$$I_c = \frac{b-a}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b) \right]. \qquad (5.6)$$

5.1.2. Обобщенные формулы прямоугольников, трапеций и Симпсона

Из рисунков 5.1- 5.5 видно, что чем больше длина отрезка [a, b], тем, как правило, больше погрешность, то есть разность между точным значением интеграла и приближенным, полученным по соответствующей квадратурной формуле. Для уменьшения этой погрешности поступают следующим образом. Отрезок [a, b] разбивается на n частей. Для простоты будем считать, что эти части равные, то есть $h_i = x_{i+1} - x_i = h = const$. На каждом из отрезков [x_i, x_{i+1}] используется соответствующая формула. Получают так называемые обобщенные формулы.

Обобщенная формула левых прямоугольников (рис. 5.6).

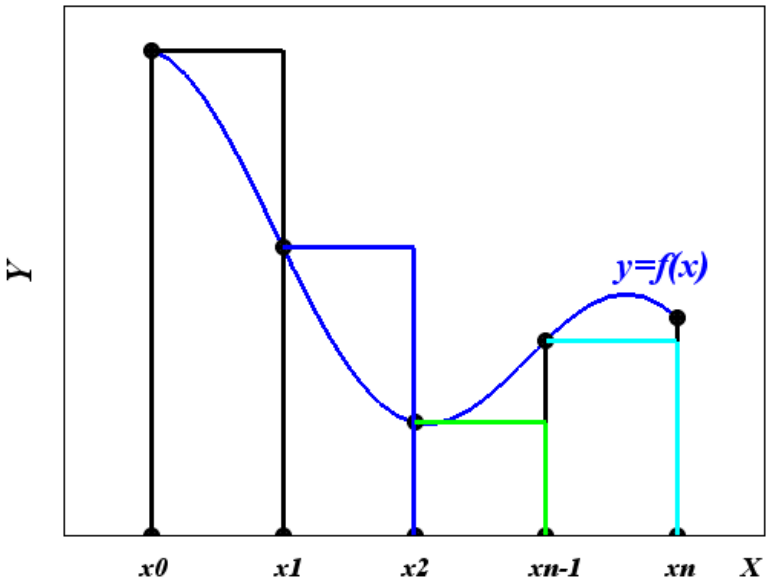


Рис. 5.6. Геометрическая интерпретация обобщенной формулы левых прямоугольнков

Если вычислить сумму площадей каждого получившегося прямоугольника, получим:

$$S = hf(x_0) + hf(x_1) + hf(x_2) + \ldots + hf(x_{n-1}) = h\sum_{i=0}^{n-1} f(x_i).$$

Здесь $x_0=a$, $x_n=b$, $x_{i+1}=x_i+h$.

Таким образом,

$$I^{o}_{лпр} = h\sum_{i=0}^{n-1} f(x_i). \qquad (5.7)$$

Эту формулу нетрудно запрограммировать.

Обобщенная формула правых прямоугольников

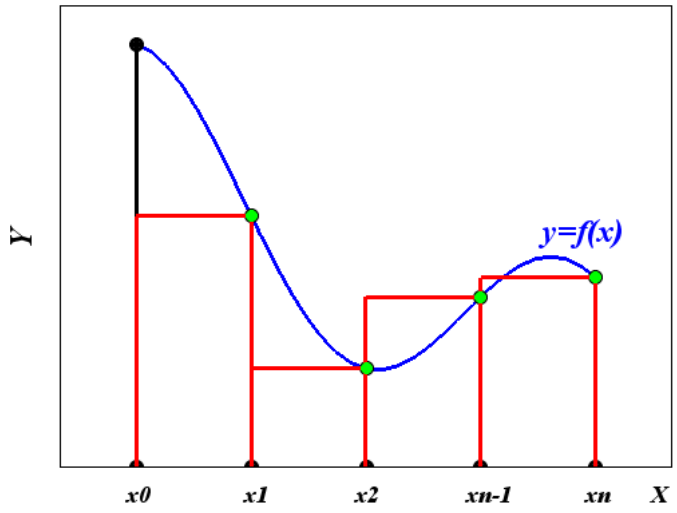


Рис. 5.7. Геометрическая интерпретация обобщенной формулы правых прямоугольнков

Если вычислить сумму площадей всех получившихся прямоугольников, получим:

$$S = hf(x_1) + hf(x_2) + hf(x_3) + \ldots + hf(x_n) = h\sum_{i=1}^{n} f(x_i).$$

Таким образом,

$$I^{o}_{npp} = h\sum_{i=1}^{n} f(x_i). \qquad (5.8)$$

Обобщенная формула центральных прямоугольников. Для ее применения нам понадобится другая таблица.

x	$x_0+h/2$	$x_1+h/2$	$x_2+h/2$	…	$x_{n-1}+h/2$
$f(x)$	$f(x_0+h/2)$	$f(x_1+h/2)$	$f(x_2+h/2)$	…	$f(x_{n-1}+h/2)$

Высотой каждого прямоугольника является значение функции в середине каждого из отрезков $[x_i, x_{i+1}]$.

Если вычислить сумму площадей всех получившихся прямоугольников, получим:

$$S = hf(x_0 + \frac{h}{2}) + hf(x_1 + \frac{h}{2}) + hf(x_2 + \frac{h}{2}) + \ldots + hf(x_{n-1} + \frac{h}{2}) = h\sum_{i=0}^{n-1} f(x_i + \frac{h}{2}).$$

Таким образом,

$$I^{o}_{цпр} = h\sum_{i=0}^{n-1} f(x_i + \frac{h}{2}). \qquad (5.9)$$

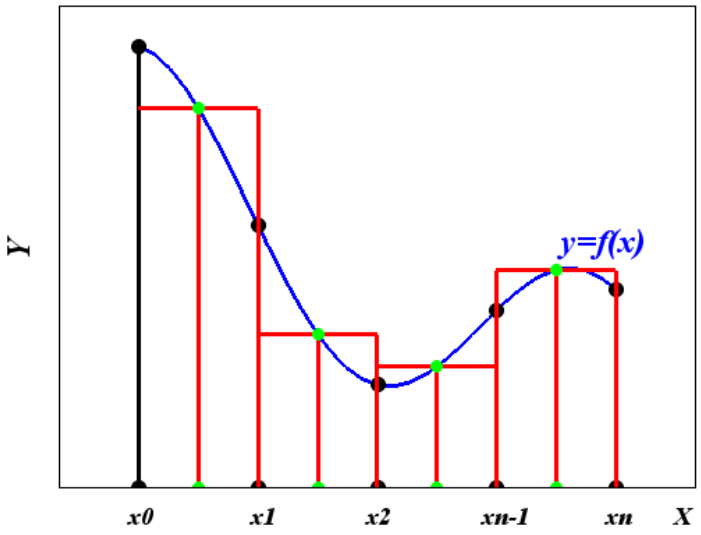


Рис. 5.8. Геометрическая интерпретация обобщенной формулы центральных прямоугольнков

Обобщенная формула трапеций.

Осуществив линейную интерполяцию подынтегральной функции на каждом из отрезков $[x_i, x_{i+1}]$, получим:

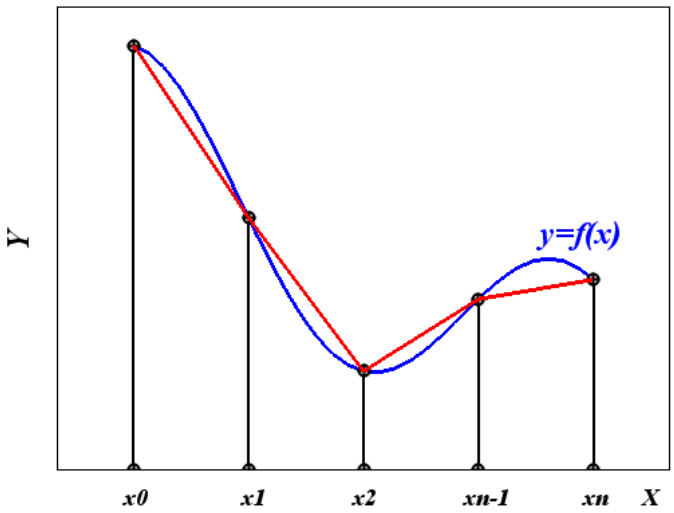


Рис. 5.9. Геометрическая интерпретация обобщенной формулы трапеций

$$I_{mp}^{o} = h\left(\frac{f(x_0) + f(x_n)}{2} + \sum_{i=1}^{n-1} f(x_i) \right). \qquad (5.10)$$

Заменив подынтегральную функцию на параболу второй степени на каждом из отрезков $[x_i, x_{i+2}]$ и вычислив сумму площадей получившихся криволинейных трапеций аналогично тому, как это было сделано на с. 73, получим обобщенную формулу Симпсона:

$$I_c^o = \frac{h}{3}\left[f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + \ldots + 2f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)\right] \quad (5.11)$$

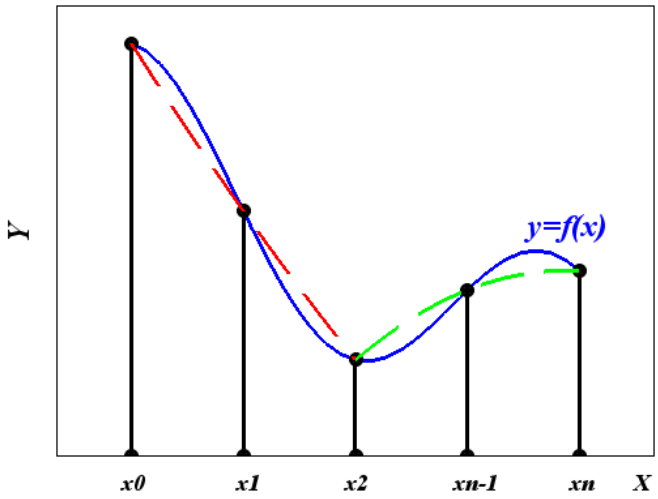


Рис. 5.10. Геометрическая интерпретация обобщенной формулы Симпсона

Первая парабола проходит через три точки с координатами (x_0, $f(x_0)$), (x_1, $f(x_1)$) и (x_2, $f(x_2)$), вторая – через точки (x_2, $f(x_2)$), (x_3, $f(x_3)$) и (x_4, $f(x_4)$), последняя – через точки (x_{n-2}, $f(x_{n-2})$), (x_{n-1}, $f(x_{n-1})$) и (x_n, $f(x_n)$). Число точек в формуле Симпсона должно быть нечетным. Чем больше n, тем, как правило, меньше погрешность $E(f)$. Формулы дают следующую погрешность. Погрешность обобщенных формул левых и правых прямоугольников не превышает величины $E(f) = \frac{|f'(\varsigma)|}{2}(b-a)h$, погрешность обобщенной формулы центральных прямоугольников – величины $E(f) = \frac{|f''(\varsigma)|}{24}(b-a)h^2$, погрешность обобщенной формулы трапеций – $E(f) = \frac{|f''(\varsigma)|}{12}(b-a)h^2$, погрешность обобщенной формулы Симпсона – не превышает $\frac{b-a}{2880}h^4\left|f^{(4)}(\varsigma)\right|$ [1]. Здесь $|f^{(n)}(\varsigma)|$ – максимальное значение модуля соответствующей производной на интервале интегрирования $[a, b]$.

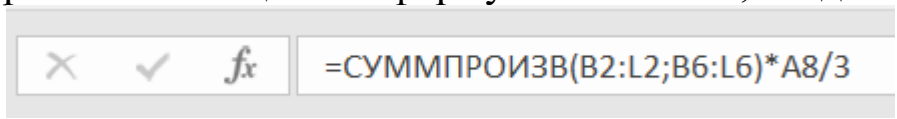
Пример 5.1. Разбив интервал интегрирования на 10 равных частей, вычислить определенный интеграл $\int_1^5 \frac{x\ln x+\sin x}{\cos x+2}dx$ по обобщенным формулам прямоугольников, трапеций и Симпсона. Составить программы вычисления этого интеграла по указанным формулам в среде Scilab.

Решение. Всю вычислительную работу проведем в Excel.

Шаг в таблице равен $h=\frac{b-a}{10}=\frac{5-1}{10}=0.4$.

	A	B	C	D	E	F	G	H	I	J	K	L
1	x	1	1,4	1,8	2,2	2,6	3	3,4	3,8	4,2	4,6	5
2	$f(x)=\frac{x\ln x+\sin x}{\cos x+2}$	0,3312	0,6712	1,1461	1,8017	2,6243	3,4029	3,7798	3,68985	3,415	3,1921	3,1039
3		1	1	1	1	1	1	1	1	1	1	0
4		0	1	1	1	1	1	1	1	1	1	1
5		0,5	1	1	1	1	1	1	1	1	1	0,5
6		1	4	2	4	2	4	2	4	2	4	1
7	h											
8	*0,4*											
9	Обобщенные формулы											
10	Левых прямоугольников	9,62168615										
11	Правых прямоугольников	10,73074789										
12	Трапеции	10,17621702										
13	Симпсона	10,18621189										

В ячейки B3-L6 введены коэффициенты соответствующих квадратурных формул. Для облегчения расчетов воспользуемся математической функцией СУММПРОИЗВ. Так, в ячейку B13 для вычисления интеграла по обобщенной формуле Симпсона, введена формула



Для вычисления интеграла по обобщенной формуле центральных прямоугольников нам понадобится другая таблица:

18	x	1,2	1,6	2	2,4	2,8	3,2	3,6	4	4,4	4,8
19	$f(x)=\frac{x\ln x+\sin x}{\cos x+2}$	0,4872	0,8888	1,4494	2,19909	3,0422	3,65747	3,77872	3,55654	3,2892	3,1297
20											
21	Обобщенная формула центральных прямоугольников	10,19124288									

В ячейку B21 введена формула

Программа на Scilab:

```
clc
function y=f(x)
   y=(x.*log(x)+sin(x))./(cos(x)+2);
endfunction
s1=0;s2=0;s3=0;
a=1;b=5;n=10;
h=(b-a)/10;i=1;
for x=a+h:h:b-h+h/100
   s1=s1+f(x);
   s2=s2+(i+3)*f(x);i=-i;
   s3=s3+f(x-h/2);
end
trap=((f(a)+f(b))/2+s1)*h;
Simps=(f(a)+f(b)+s2)*h/3;
leftp=(s1+f(a))*h;
rightp=(s1+f(b))*h;
centp=(s3+f(b-h/2))*h;
printf("Интеграл по обобщенной формуле левых прямоугольников равен %2.6f",leftp);disp("")
printf("Интеграл по обобщенной формуле правых прямоугольников равен %2.6f",rightp);disp("")
printf("Интеграл по обобщенной формуле центральных прямоугольников равен %2.6f",centp);
disp("")
printf("Интеграл по обобщенной формуле трапеций равен %2.6f",trap);disp("")
printf("Интеграл по обобщенной формуле Симпсона равен %2.6f",Simps);disp("")
```

Результаты:

Интеграл по обобщенной формуле левых прямоугольников равен 9.621686

Интеграл по обобщенной формуле правых прямоугольников равен 10.730748

Интеграл по обобщенной формуле центральных прямоугольников равен 10.191243

Интеграл по обобщенной формуле трапеций равен 10.176217

Интеграл по обобщенной формуле Симпсона равен 10.186212

5.2. Решение задачи численного интегрирования в Scilab

При вычислении определенных интегралов от функций, заданных в виде таблицы или в явном виде ($\int_a^b f(x)dx$), одним из численных методов используют функции *intsplin, inttrap, integrate, intg*.

Вычислим интеграл из примера 5.1 с помощью команды *intsplin*. Это интегрирование экспериментальных данных с помощью сплайн-интерполяции. Известно значение интегрируемой функции в дискретных точках (узлах).

```
-->x=1:.4:5;
-->y=(x.*log(x)+sin(x))./(cos(x)+2);
-->v=intsplin(x,y)
```

Получаем:

```
 v  =    10.186009
```

Или:

```
-->x=[1 1.4 1.8 2.2 2.6 3 3.4 3.8 4.2 4.6 5];
-->y=[0.3312484  0.6712133  1.1461338  1.8017036  2.6242688
3.4029023  3.7797992  3.6898487  3.4150132  3.1920841
3.1039027];
-->v=intsplin(x,y)
 v  =    10.186009
```

Тот же интеграл вычислим с помощью функции *inttrap*. Интегрирование экспериментальных данных производится по формуле трапеций. Задав те же самые *x* и *y*, получаем:

```
 v=inttrap(x,y)
 v  =    10.176217
```

Остальные функции используют, когда подынтегральная функция задана явно.

Функция *integrate*. Это интегрирование по квадратуре. Может задаваться требуемая точность вычислений.

Синтаксис функции: *[x]=integrate(f,v,a,b [,atol [,rtol]])*

Параметры функции:

f – подынтегральная функция;

v – переменная интегрирования;

a, b – пределы интегрирования;

atol, rtol – действительные числа, абсолютная и относительная ошибки соответственно. Первая по умолчанию равна 10^{-8}, вторая 10^{-14}.

Ниже показано, как можно вычислить интеграл с помощью данной функции.

*--> integrate('(x*log(x)+sin(x))/(cos(x)+2)','x',1,5)*

ans = 10.186238

Можно это сделать и так:

```
clc
function y=f(x),    y=(x.*log(x)+sin(x))./(cos(x)+2);endfunction
t=integrate('f','x',1,5);printf("Интеграл равен %2.6f",t);disp("")
```

Получаем:

Интеграл равен 10.186238

Функция *intg* имеет такой синтаксис:

[v, err [,ierr]]=intg(a, b, f [,ea [,er]])

Параметры функции:

a, b – пределы интегрирования;

ea, er – действительные числа, абсолютная и относительная ошибки соответственно. По умолчанию равны 10^{-13} и 10^{-8}.

err – оценка абсолютной ошибки результата.

Вот код с использованием этой функции для вычисления интеграла из примера 5.1.

```
clc
function y=f(x),    y=(x.*log(x)+sin(x))./(cos(x)+2);endfunction
t=intg(1,5,f);printf("Интеграл равен %2.6f",t);disp("")
```

Результат:

Интеграл равен 10.186238

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

Задание 1. Взяв шаг равным одной десятой длины интервала интегрирования, вычислить определенный интеграл

$$a)\ \int_{\nu+0.2\mu-1}^{2+1.3\nu} e^{\frac{(x-\nu)^2}{8}}\,dx \qquad b)\ \int_{\nu+0.1\mu}^{5+1.3\nu} \sqrt{x^2+\nu+\gamma}\,dx$$

по обобщенным формулам прямоугольников, трапеций и Симпсона. Тот же интеграл вычислить в среде Scilab с помощью одной из приведенных в пособии специальных функций.

Задание 2. Составить программу для вычисления интеграла из задания 1 по обобщенным формулам прямоугольников, трапеций и Симпсона.

Тема 6. Численные методы решения обыкновенных дифференциальных уравнений

ОДУ порядка n называется уравнение вида

$$F(x, y(x), y'(x), y''(x), ..., y^{(n)}(x)) = 0 \quad (6.1)$$

где n - порядок наивысшей производной, входящей в уравнение.

Примеры дифференциальных уравнений:

$$2\frac{d^2y}{dx^2} + 3\frac{dy}{dx} + 4y = 1 \quad (6.2)$$

$$\left(\frac{dy}{dx}\right)^2 - xy^4\frac{dy}{dx} + \sin y = 0 \quad (6.3)$$

Здесь $y(x)$ – неизвестная функция.

Уравнение (6.2) имеет порядок 2, уравнение (6.3) – порядок 1.

Если уравнение линейно по $y(x), y'(x), y''(x), ..., y^{(n)}(x)$, то оно называется линейным. (6.2) – линейное уравнение, (6.3) – нелинейное.

Запишем дифференциальное уравнение n порядка в явном виде:

$$y^{(n)}(x) = f(x, y(x), y'(x), y''(x), ..., y^{(n-1)}(x)) \quad (6.4)$$

Уравнение в виде (6.1) – уравнение в неявной форме.

Под интегрированием уравнения (6.1) понимают нахождение функции $y(x)$, которая удовлетворяет этому уравнению. $y(x)$ называется решением дифференциального уравнения. Общее решение ОДУ n -го порядка имеет вид:

$$y = y(x, c_1, c_2, ..., c_n) \quad (6.5)$$

где $c_1, c_2, ..., c_n$ - произвольные константы.

При любом наборе конкретных констант получаются частные решения.

Задача Коши есть задача о нахождении частного решения уравнения (6.4), удовлетворяющего начальным условиям

$$y(x_0) = y_0,\ y'(x_0) = y_0',\ ...,\ y^{(n-1)}(x_0) = y_0^{(n-1)} \quad (6.6)$$

Здесь $y_0, y_0', ..., y_0^{(n-1)}$ - некоторые заданные числа.

Графическое изображение частного решения называют интегральной кривой. Общее решение дифференциального уравнения n-го порядка определяет n-параметрическое семейство интегральных кривых.

6.1. Задача Коши для обыкновенного дифференциального уравнения 1-го порядка

Рассмотрим два численных метода решения ОДУ первого порядка. Пусть дифференциальное уравнение первого порядка задано в виде

$$\begin{cases} y' = f(x,y) \\ y(x_0) = y_0 \end{cases} \tag{6.7}$$

Задача (6.7) – задача Коши для ОДУ первого порядка.

Пример:

$y' = x$ Общее решение этого уравнения $y(x) = \frac{x^2}{2} + c$

$$\begin{cases} y' = x \\ y(2) = 4 \end{cases}$$

Решением этой задачи Коши будет единственная функция

$$y(x) = \frac{x^2}{2} + 2$$

Так решались ДУ в курсе высшей математики.

Метод Эйлера

Перепишем уравнение (6.7) в виде

$$y' = \lim_{\Delta x \to 0} \frac{\Delta y}{\Delta x} = f(x,y)$$

$\frac{y(x_{i+1}) - y(x_i)}{x_{i+1} - x_i} \approx f(x_i, y_i)$ Обозначим $x_{i+1} - x_i = h_i$. Тогда

$$y(x_{i+1}) = y(x_i) + h_i f(x_i, y_i) \tag{6.8}$$

Формула (5.8) – формула метода Эйлера решения ОДУ первого порядка.

$y_1 = y(x_1) = y(x_0) + h_0 f(x_0, y_0)$

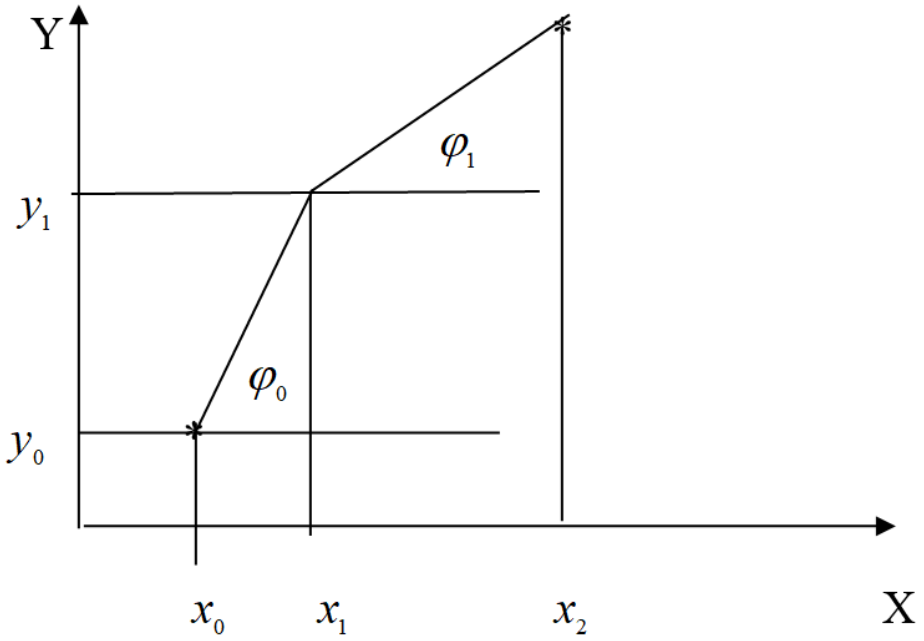
$x_1 = x_0 + h_0$

$y'(x_0) = f(x_0, y_0) = tg\varphi_0$

$y_2 = y(x_2) = y(x_1) + h_1 f(x_1, y_1)$

$x_2 = x_1 + h_1$

$y'(x_1) = f(x_1, y_1) = tg\varphi_1$ и т.д.



Результатом численного решения дифференциального уравнения является таблица

X	x_0	x_1	x_2	…	x_n
Y	y_0	y_1	y_2	…	y_n

где x_0 *и* y_0 заданы, $x_{i+1} = x_i + h_i$ (шаг h_i задается и может быть как переменным, так и постоянным, а y_i ($i=1, 2, \ldots, n$) вычисляются по соответствующим формулам (например, по формуле (6.8)).

Пример:

$$\begin{cases} y' = x + y \\ y(0) = 0 \end{cases}$$ h=0.2 Вот точное решение: $y^T = e^x - x - 1$

$$y_1 = y(x_1) = y(x_0) + hf(x_0, y_0)$$

$$x_0 = 0,\ y_o = 0,\ h = 0.2,\ f(x, y) = x + y$$

$$y_1 = 0 + 0.2(0 + 0) = 0$$

$$y_2 = y(x_2) = y(x_1) + hf(x_1, y_1) = 0 + 0.2(0.2 + 0) = 0.04$$

$$x_1 = x_0 + h = 0 + 0.2 = 0.2$$

$$y_3 = y(x_3) = y(x_2) + hf(x_2, y_2) = 0.04 + 0.2(0.4 + 0.04) = 0.128$$

$$x_2 = x_1 + h = 0.4$$

Сведем результаты в таблицу:

x	$y^э$	$y^T = e^x - x - 1$	$y^{р-к}$
0	0	0	0
0.2	0	0.021403	0.0214
0.4	0.04	0.091825	
0.6	0.12	0.222119	

Погрешность метода Эйлера – o(h).

Метод Рунге-Кутта решения ОДУ первого порядка

$$\begin{cases} y' = f(x, y) \\ y(x_0) = y_0 \end{cases}$$

$$y_{i+1} = y_i + \frac{k_{1i} + 2k_{2i} + 2k_{3i} + k_{4i}}{6}$$

$$k_{1i} = h_i f(x_i, y_i)$$

$$k_{2i} = h_i f(x_i + \frac{h_i}{2}, y_i + \frac{k_{1i}}{2})$$

$$k_{3i} = h_i f(x_i + \frac{h_i}{2}, y_i + \frac{k_{2i}}{2})$$

$$k_{4i} = h_i f(x_i + h_i, y_i + k_{3i})$$

Погрешность метода Рунге-Кутта – $o(h^4)$.

Пример:

$$\begin{cases} y' = x + y \\ y(0) = 0 \end{cases} \quad \text{h=0.2} \quad y^T = e^x - x - 1$$

$$y_1 = y_0 + \frac{k_{10} + 2k_{20} + 2k_{30} + k_{40}}{6}$$

$$k_{10} = hf(x_0, y_0) = 0.2 f(0,0) = 0.2(0 + 0) = 0$$

$$k_{20} = hf(x_0 + \frac{h}{2}, y_0 + \frac{k_{10}}{2}) = hf(0.1,0) = 0.2(0.1 + 0) = 0.02$$

$$k_{30} = hf(x_0 + \frac{h}{2}, y_0 + \frac{k_{20}}{2}) = hf(0.1,0.01) = 0.2(0.1 + 0.01) = 0.022$$

$$k_{40} = hf(x_0 + h, y_0 + k_{30}) = hf(0.2,0.022) = 0.2(0.2 + 0.022) = 0.0444$$

$$y(x_1) = 0 + \frac{0 + 2 * 0.02 + 2 * 0.022 + 0.0444}{6} = 0.0214$$

6.2. Решение дифференциальных уравнений первого порядка и систем дифференциальных уравнений первого порядка в Scilab

Для того чтобы решить обыкновенное дифференциальное уравнение вида $y' = f(x, y), y(x_0) = y_0$, можно воспользоваться функцией y=ode(y0,x0,x,f).

Здесь y0 - вектор начальных условий; x0 - начальная точка интервала интегрирования ; x - координаты узлов сетки, в которых происходит поиск решения; f - функция, определяющая правую часть уравнения или системы уравнений. y - вектор решений.

Рассмотрим работу функции на примере следующих задач.

ЗАДАЧА.

Решить задачу Коши

$y' + y = \sin(xy), \quad y(0) = 1.5$

Решение

Перепишем уравнение следующим образом: $y' = -y + \sin(xy), y(0) = 1.5$

Создадим функцию, описывающую правую часть уравнения, и применим функцию z=ode(y0,x0,x,f), в качестве параметров которой будем использовать следующие:

f - предварительно созданная функция *f*(*x*, *y*) (*f(x,y)=-y+sin(xy)* – правая часть нашего дифуравнения);

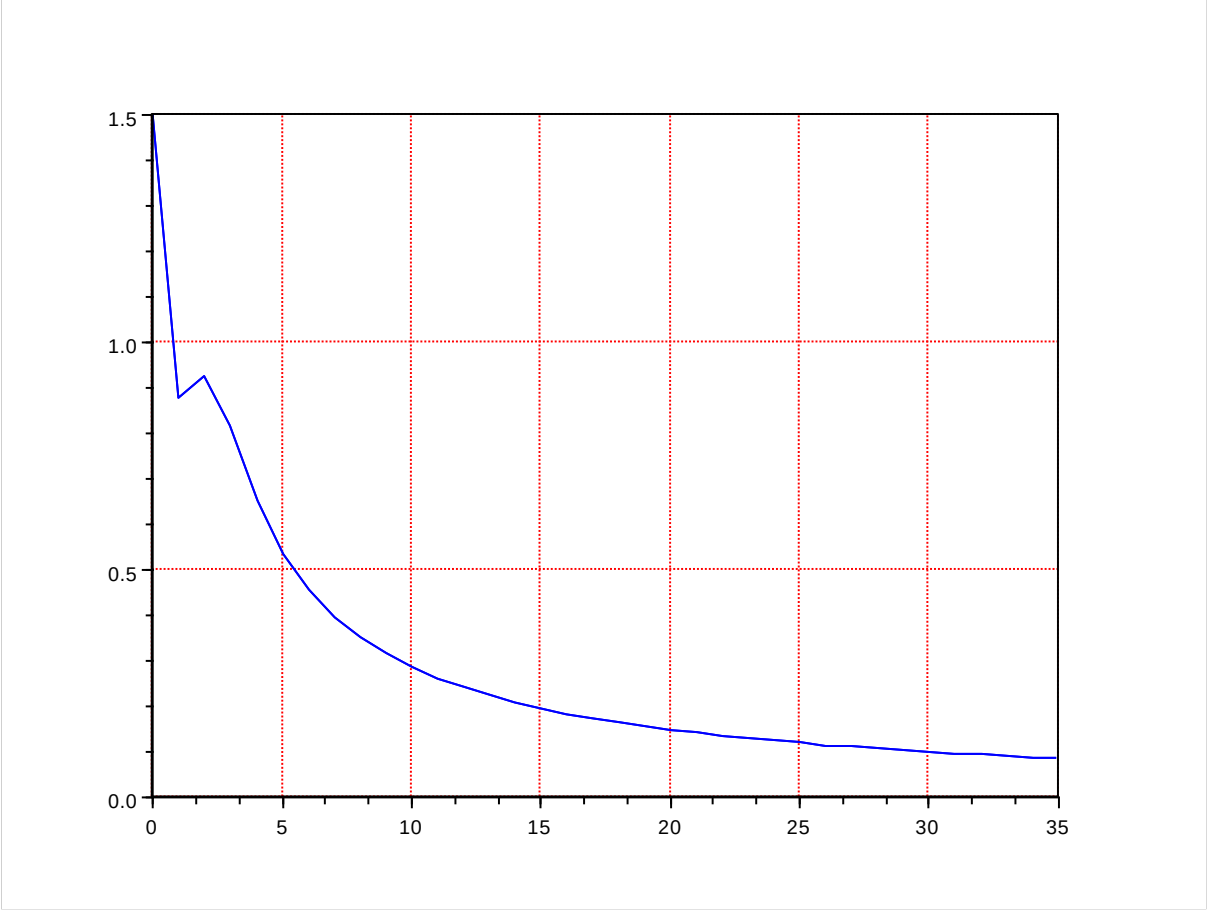
x - координаты сетки;

y0, x0 - начальное условие *y*(0)=1.5;

z - результат работы функции.

Приведем код программы, которая строит график, моделирующий процесс, описанный данным уравнением. Здесь x меняется от 0 до 35 с шагом 1. Решение уравнения в точках 0, 1,2,…,35 задается вектором z, координаты которого можно распечатать, убрав точку с запятой в конце 5 строки.

```
clc
clf()
function w=f(x,y),w=-y+sin(x*y),endfunction;
x0=0;y0=1.5;x=0:1:35;
z=ode(y0,x0,x,f);
disp([x;z])
plot(x,z);xgrid(5)
```



Графическое решение дифференциального уравнения

На экран будут выведены 36 пар значений x и z.

Заменив пятую строку на *z=ode("rkf",y0,x0,x,f)*, получим решение уравнения методом Рунге-Кутта.

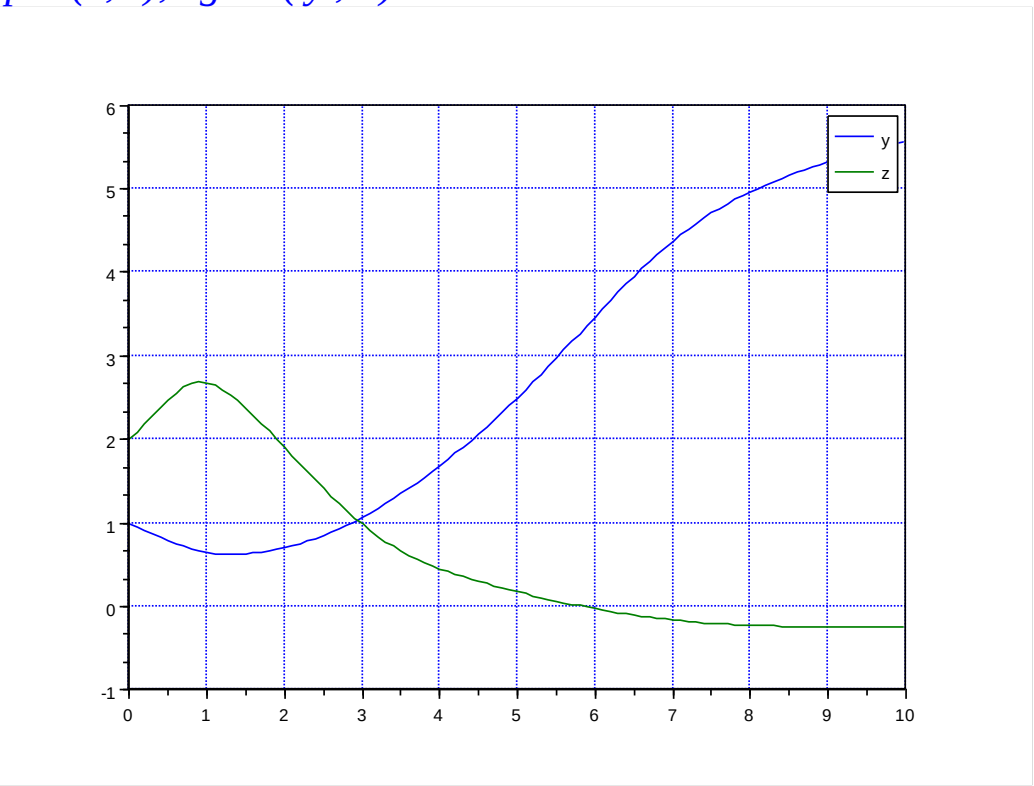
Пример. Решить задачу Коши для системы двух дифференциальных уравнений первого порядка

$$\begin{cases} y' = \cos(yz) \\ z' = \sin(y + xz) \\ y(0) = 1 \\ z(0) = 2 \end{cases}$$

на интервале [0; 10].

Приведем функцию, описывающую заданную систему обыкновенных дифференциальных уравнений, и команды Scilab, необходимые для ее численного и графического решения. Вместо переменной y в программе использована переменная w(1), вместо переменной z – переменная w(2).

```
clf()
//Функция, описывающая систему дифференциальных уравнений
function g=syst(x,w)
g=zeros(2,1);g(1)=cos(w(1)*w(2));g(2)=sin(w(1)+w(2)*x);
endfunction
//Решение системы дифференциальных уравнений
w0=[1;2];x0=0;x=0:.1:10;w=ode(w0,x0,x,syst);xgrid(2)
//Формирование графического решения
plot(x,w),legend('y','z')
```



Графическое решение задачи

В программе *zeros(2,1)* матрица, состоящая из нулей, размерностью 2x1.

```
-->g=zeros(2,1)
 g  =
    0.
    0.
```

Первый аргумент вектора w - это y(0), второй z(0).

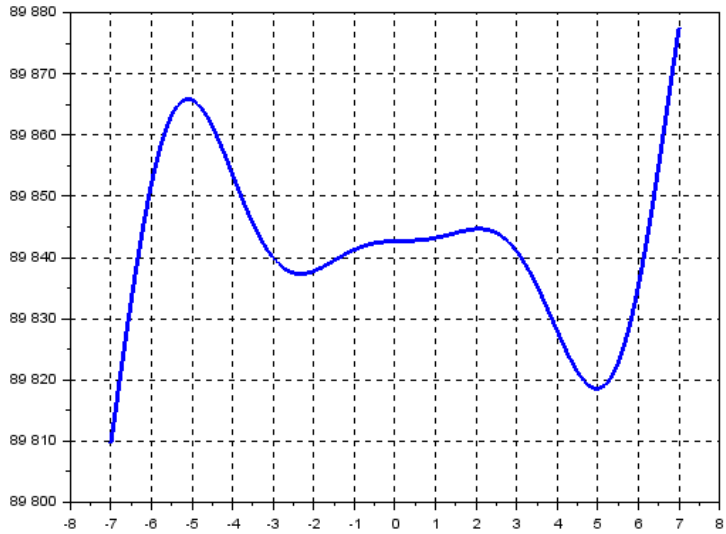
При необходимости решение w можно вывести на экран. Например, это можно сделать так: disp([x;w])

Тема 7
Методы одномерной оптимизации

На практике часто возникает задача поиска экстремумов некоторой целевой функции $y = f(x_1, x_2, ..., x_n)$ n переменных. Такая функция описывает n+1 – мерную поверхность. Соответственно функция $y = f(x)$ одной переменной описывает некоторую кривую на плоскости.

Поиск экстремумов функции одной переменной является самостоятельной и часто встречаемой задачей. Кроме того, к нему сводится гораздо более сложная задача поиска экстремумов функции многих переменных.

В общем случае функция $y = f(x)$ может иметь несколько экстремумов – максимумов или минимумов.



Из них главный называется глобальным. Задача поиска экстремумов сводится к их локализации и уточнению значений x^* и $f(x^*)$ в точке экстремума. В дальнейшем для функций одной переменной под экстремумом будем подразумевать минимум функции $y = f(x)$ (так как поиск максимума функции $y = f(x)$ равносилен поиску минимума функции $y = -f(x)$). Будем также полагать, что на изменения x накладываются ограничения вида $a \le x \le b$, где a и b – границы интервала поиска. В пределах отрезка [a, b] функцию считаем унимодальной, то есть содержащей один минимум.

Остановимся на двух алгоритмах поиска минимума функции одной переменной.

7.1. Метод дихотомии (деления интервала поиска [a, b] пополам).

Реализуется следующим алгоритмом. Пусть ε заданная погрешность вычисления $x_{\min}$ и параметр δ **(δ≤ε).**

1. Проверяем условие $\frac{b-a}{2} < \varepsilon$. Если это условие выполняется, идем к п. 6, если нет, то к п. 2.

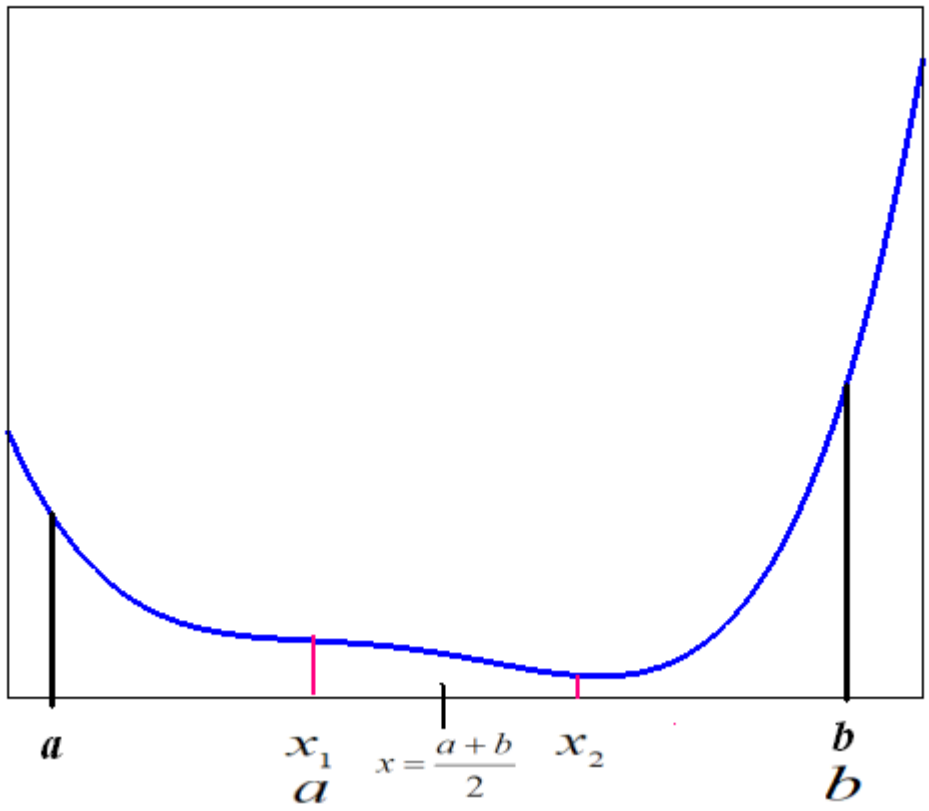
2. Делим отрезок пополам и вычисляем две абсциссы, симметрично расположенные относительно точки $x = \frac{a+b}{2}$: $x_1 = \frac{a+b-\delta}{2}$ и $x_2 = \frac{a+b+\delta}{2}$

3. Для этих значений вычисляем $f(x_1)$ и $f(x_2)$.

4. Проверяем условие $f(x_1) < f(x_2)$. Если оно выполняется, полагаем $b = x_2$ и идем к п. 1.

5. Если $f(x_1) \geq f(x_2)$, то $a = x_1$ и идем к п. 1.

6. $x_{\min} = \frac{a+b}{2}$, $y_{\min} = f(x_{\min})$.



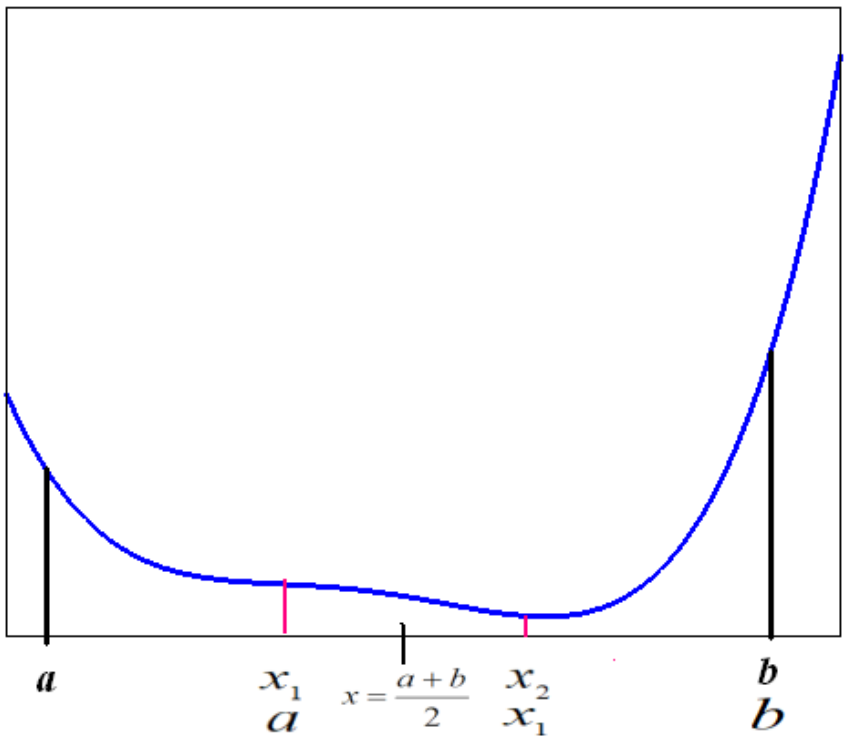
7.2. Метод золотого сечения

Основан на делении отрезка **[a, b]** по правилу золотого сечения. Он позволяет сужать отрезок **[a, b],** каждый раз вычисляя лишь одно значение $f(x)$, а не два, как в методе дихотомии. Реализуется следующим алгоритмом.

1. Находим коэффициент дробления отрезка **[a, b] :**

$$k = \frac{\sqrt{5}-1}{2} \approx 0{,}62$$

2. Находим $x_1 = a + (1-k)(b-a)$ и вычисляем $f(x_1)$.
3. Находим $x_2 = a + k(b-a)$ и вычисляем $f(x_2)$.
4. Проверяем выполнение условия $\frac{b-a}{2} < \varepsilon$. Если оно выполняется, то полагаем $x_{\min} = \frac{a+b}{2}, y_{\min} = f(x_{\min}),$ после чего останавливаем счет. Если данное условие не выполняется, идем к пункту 5.
5. Проверяем условие $f(x_1) > f(x_2)$. Если оно выполняется, то a=x_1, x_1=x_2, $f(x_1)$=$f(x_2)$, после чего выполняются пункты 3 и 4.
6. Если f(x1)≤f(x2), то b=x2, x2=x1; f(x2)=f(x1), после чего выполняются пункты 2 и 4.



Оба этих алгоритма нетрудно запрограммировать.

Подавляющее число реальных задач оптимизации, возникающих на практике, являются многомерными. В них целевая функция зависит от нескольких аргументов. Математическая постановка задач многомерной оптимизации аналогична их постановке в одномерном случае: требуется найти наибольшее (наименьшее) значение целевой функции $y = f(x_1, x_2, ..., x_n)$ **,** заданной на некотором множестве $D \subset R^n$ возможных значений ее аргументов.

Методами решения таких задач посвящены специальные разделы математики.

7.3. Решение задач оптимизации в среде Scilab

7.3.1 Функция optim

Для решения задач безусловной оптимизации в Scilab предназначена функция *optim*, которая возвращает локальный минимум целевой функции *costf* и точку *xopt*, в которой этот минимум достигается.

Для использования функции *optim* необходимо определить целевую функцию, которая в случае, если она определена в Scilab, имеет следующую структуру:

```
function [f,g,ind]=costf(x,ind)
f=gg(x);//функция, минимум которой мы ищем
g= numderivative(gg,x); //градиент функции f
endfunction
```

Переменная ind используется как входной и выходной аргумент и определяет настройку вычисления целевой функции. Если ind равен 2, 3 или 4, то функция должна вычислять только f, g и f или только g. Если ind=0, то функция costf не вычисляет ничего и используется только для вывода.

Синтаксис самой функции *optim*:

```
[fopt, xopt, [gopt, work, iters, evals, err], ti, td] = optim(costf
[,<contr>] ,x0 [,algo] [,df0 [,mem]] [,work] [,<stop>] [,<params>]
[,iprint=iflag])
```

Входными аргументами функции *optim* являются:

costf – целевая функция;

<contr> - необязательный параметр, содержащий ограничения на x: $binf \leq x_i \leq bsup$. Если эти границы задаются, то binf и bsup являются векторами той же размерности, что и x0;

x0 – вектор начальных приближений;

algo – строка, с помощью которой задается алгоритм оптимизации ("qn" – квази-ньютоновский, "gc" – сопряженных градиентов или "nd" – без дифференцирования). По умолчанию значение algo="qn";

df0 – скаляр, предположение об убывании целевой функции на первой итерации. По умолчанию df0 = 1;

mem – целое число. Это количество переменных, используемых для аппроксимации гессиана (Hessian). По умолчанию установлено значение, равное 10. Опция доступна для безусловной оптимизации при значениях параметра algo "gc" или "nd";

<stop> - совокупность переменных, контролирующих сходимость алгоритма. Можно задать значения параметров nap, iter, epsg, epsf, epsx:

nap – максимальное число вызовов функции costf (по умолчанию 100);

iter – максимальное количество итераций (по умолчанию iter=100;

epsg – ограничение на норму градиента (по умолчанию epsg= %eps=2.220D-16;

epsf – пороговое значение, контролирующее убывание f (по умолчанию epsf=0);

epsx – пороговое значение, контролирующее разброс x. Это вектор (возможно, матрица) той же размерности, что и x0. Может быть использован для масштабирования;

<params> – в случае, когда целевая функция является подпрограммой на C или Fortran, последовательность аргументов задает порядок обращения к целевой функции. Данный параметр не имеет смысла, когда целевая функция задана в Scilab;

"iprint=iflag" – аргумент задает режим трассировки. По умолчанию iprint=0: не печатается никаких дополнительных сообщений. В случае, если значение аргумента не меньше единицы,

на экран выводится больше информации. Вид и количество информации определяется типом используемого алгоритма: ("qn","gc" или "nd");

Выходные аргументы функции:

fopt – значение целевой функции в точке xopt;

xopt – найденное наилучшее значение x (минимум функции);

gopt – значение градиента целевой функции в точке xopt;

work – рабочий массив для работы квази-ньютоновского метода оптимизации. Этот массив автоматически создается, когда происходит вызов функции optim. Он может быть задан как входной параметр функции для ускорения расчетов;

iters – скаляр. Число итераций, выводимых на печать, при значении iprint=2;

evals - скаляр. Число вызовов функции, выводимых на печать, при значении iprint=2;

err – целое число, индикатор успешности решения задачи. Принимает значения от 1 до 10. Если задача решена успешно, err=9.

Функция предназначена для нелинейной безусловной оптимизации функции *f(x)*. *x* – это вектор.

С помощью этой функции можно решать задачи вида:

min f(x)

binf <= x <= bsup

Аргумент *costf* может быть функцией Scilab, списком или строкой, задающей имя подпрограммы на C или Fortran. Эта внешняя функция должна выдавать значение целевой функции *f* в точке *x*, а также значение градиента (производной) целевой функции в этой точке.

Пример 7.1. Найти минимум функции $y = x^4 - 3x^2 - 5x - 4.$

Решение. Построим график функции для определения интервалов [*a*, *b*], на которых находятся экстремумы этой функции.

```
clf()
clc
function y=gg(x)
  y=x.^4-3*x.^2-5*x-4
endfunction
x=-3:.01:4;
plot(x,gg(x));xgrid(5)
gca.children.children.thickness=3;
```

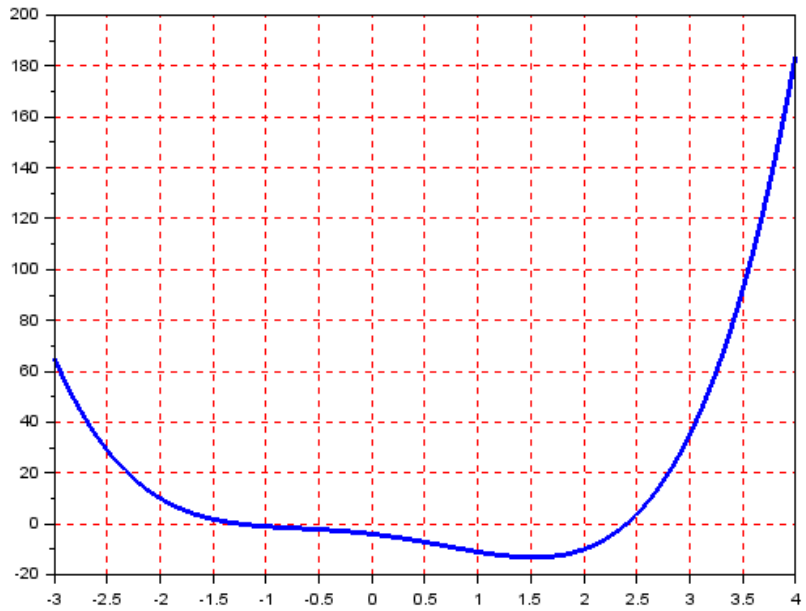


Рис. 7.1. График функции $y=x^4-3x^2-5x-4$

Из графика видно, что минимум – один, и он находится на отрезке [1, 2]. Дополним приведенную выше программу:

```
function [f, g, r]=z(x, r)
  f=gg(x)
  g=numderivative(gg,x)
endfunction
x0=1;
[fmin,xmin,gopt]=optim(z,x0) ;disp([fmin,xmin,gopt])
```

Запустив полученную программу, получаем:

```
-13.193373   1.5233402  -9.628D-11
```

Точка минимума – xmin=1.5233402, значение целевой функции в этой точке равно fmin= -13.193373. Значение градиента в этой точке – gopt=-9.628D-11. Производная в точке минимума практически равна нулю, что говорит о правильности решения задачи.

Решим теперь эту же задачу с помощью алгоритма "nd" (без дифференцирования).

```
clc
[fopt, xopt] = optim(z, x0, "nd");disp([fopt, xopt])
```

Получаем следующие значения целевой функции и аргумента: -13.070679 1.6263555.

Решение получилось хуже, чем в первом случае.

Найдем теперь минимум этой функции на отрезке [-3, 1].

```
clc;[fopt, xopt] = optim(z, "b", -3,1, x0);disp([fopt, xopt])
```

Точка минимума находится на правом конце отрезка, xmin=1, значение целевой функции в этой точке равно 11.

Покажем, как решается задача поиска экстремумов функции нескольких переменных.

Пример 7.2. Найти точку минимума функции двух переменных $y = x_1^3 - 2x_1x_2 + x_2^2 - 3x_1 - 2x_2$.

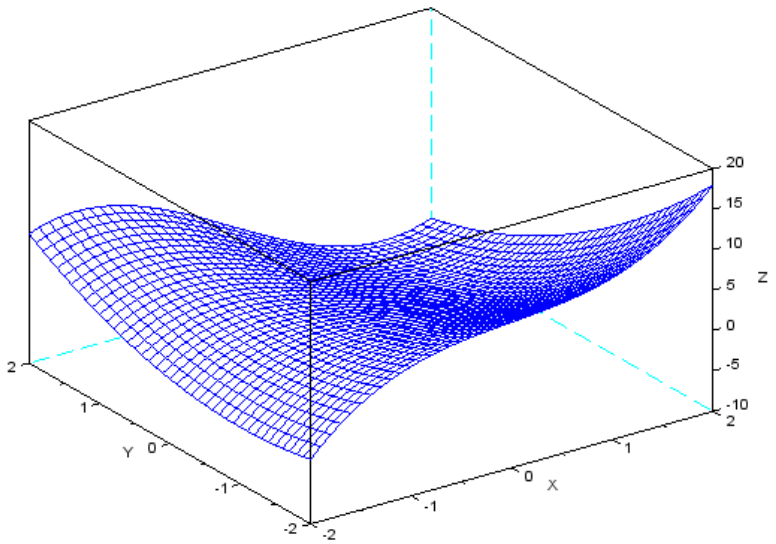


Рис. 7.2. График функции $y = x_1^3 - 2x_1x_2 + x_2^2 - 3x_1 - 2x_2$.

Ниже приведена программа поиска минимума данной функции.

```
clc
x0=[2;2];
function y=gg(x),y=x(1)^3-2*x(1)*x(2)+x(2)^2-3*x(1)-2*x(2)
endfunction
function [f, g, ind]=cst(x, ind),
f=gg(x);g=numderivative(gg,x);endfunction
[f,xopt]=optim(cst,x0)
disp(xopt,"Точка минимума " );
disp(f,"Значение функции в точке минимума ");
```

```
Точка минимума
  1.6666667
  2.6666667

 Значение функции в точке минимума

 -7.4814815
```

7.3.2. Решение задач линейного программирования. Функция karmarkar

Задачи линейного программирования – это задачи многомерной оптимизации с ограничениями в случае, когда целевая функция линейна относительно своих аргументов и все ограничения на эти аргументы имеют вид линейных уравнений и/или неравенств:

$$\min c^T \cdot x$$

$$A_{eq} x = b_{eq}$$

$$Ax \le b$$

$$l_b \le x \le u_b$$

Для решения задачи в Scilab используют функцию *karmarkar*.

Вот один из вариантов этой функции:

[xopt,fopt,exitflag,iter,yopt] =karmarkar(Aeq, beq,c,x0,rtolf, gam, maxiter,outfun,A,b,lb,ub)

Аргументы функции:

Aeq – матрица размерности *n*x*p*, где *n* – число ограничений-равенств, *p* – число неизвестных. Это матрица системы линейных ограничений;

beq – вектор-столбец размерности *n*, правая часть системы ограничений-равенств;

c – вектор-столбец размерности *p*, вектор коэффициентов целевой функции;

x0 – вектор-столбец размерности *p*, вектор начальных приближений. По умолчанию x0=[]. Если x0==[], функция *karmarkar* автоматически задает начальное приближение;

rtolf – скаляр, относительная погрешность вычисления f(x)=c'*x (по умолчанию rtolf=10^{-5});

gam – скаляр, коэффициент масштабирования. По умолчанию gam=0.5. 0<gam<1. Иногда установленное по умолчанию значение параметра gam приводит к тому, что алгоритм не обеспечивает сходимость к точке минимума. В этом случае можно попробовать уменьшить значение gam;

maxiter – скаляр, задающий максимальное число итераций (по умолчанию maxiter=200;

outfun – функция или список, функция вывода (см. Help);

A – матрица ограничений-неравенств размерности *ni*x*p*;

b – вектор-столбец, правая часть системы линейных неравенств;

lb – вектор-столбец, нижние границы переменной *x*;

ub – вектор-столбец, верхние границы переменной *x*;

Выходные аргументы функции *karmarkar*:

xopt – вектор-столбец размерности *p*, оптимальное решение;

fopt – скаляр, значение целевой функции в точке xopt;

exitflag – скаляр, статус исполнения (см. Help);

iter – скаляр, число итераций;

yopt – структура, состоящая из четырех полей, содержащих множители Лагранжа для неравенств (yopt.ineqlin), ограничений-

равенств (yopt.eqlin), верхних (yopt.upper) и нижних (yopt.lower) границ;

Если в решаемой задаче нет ограничений-неравенств (то есть (A==[] & lb==[] & ub==[]), функция *karmarkar* гарантированно возвращает вектор xopt, состоящий из неотрицательных значений.

Если заданы какие-либо ограничения-неравенства (то есть (A==[] & lb==[] & ub==[])) , то функция не обязательно возвращает неотрицательные значения переменной *xopt*.

Если в задаче отсутствуют ограничения-неравенства, то решается задача:

$$\min c^T \cdot x$$
$$A_{eq} x = b_{eq}$$
$$x \geq 0$$

Если в задаче присутствуют ограничения-неравенства, то решается задача:

$$\min c^T \cdot x$$
$$A_{eq} x = b_{eq}$$
$$Ax \leq b$$
$$l_b \leq x \leq u_b$$

Для демонстрации предназначения некоторых параметров решим задачу с тремя неизвестными и двумя ограничениями-равенствами:

$$\min -x_1 - x_2$$
$$x_1 - x_2 = 0$$
$$x_1 + x_2 + x_3 = 2$$
$$x \geq 0$$

Код программы:

```
clc
Aeq = [1 -1 0;1  1 1];
beq = [0;2];
```

```
c = [-1;-1;0];
x0 = [0.1;0.1;1.8];
[xopt,fopt,exitflag,iter,yopt]=karmarkar(Aeq,beq,c)
xstar=[1 1 0]'
```

Результат:

```
--> [xopt,fopt,exitflag,iter,yopt]=karmarkar(Aeq,beq,c)
 yopt  =
   ineqlin: [0x0 constant]
   eqlin: [2x1 constant]
   lower: [3x1 constant]
   upper: [3x1 constant]
 iter  =    68.
 exitflag  =    1.
 fopt  =   -1.9999898
 xopt  =
   0.9999949
   0.9999949
   0.0000102
```

При необходимости мы можем узнать значения множителей Лагранжа:

```
-->yopt.ineqlin
 ans  =
     []
-->yopt.eqlin
 ans  =
  - 6.483D-17
    1.
-->yopt.lower
 ans  =
  - 2.070D-10
```

```
 - 2.070D-10
   1.
-->yopt.upper
 ans  =
   0.
   0.
   0.
```

В случае, когда нам нужна большая точность, мы можем уменьшить относительную погрешность, что приведет к увеличению количество итераций.

```
-->[xopt,fopt,exitflag,iter]=karmarkar(Aeq,beq,c,[],1.e-9);
-->disp([fopt iter])
 - 2.    79.
```

Покажем, как решается задача линейного программирования, когда в ней отсутствуют ограничения-равенства.

$$\min -20x_1 - 24x_2$$

$$3x_1 + 6x_2 \le 60$$

$$4x_1 + 2x_2 \le 32$$

$$x \ge 0$$

```
clc
c = [-20 -24]';A = [3 6;4 2];b = [60 32]';
xopt=karmarkar([],[],c,[],[],[],[],A,b);disp(xopt)
```

Решение:

```
3.9999125
7.9999912
```

7.3.3. Решение задач квадратичной оптимизации. Функция qld

Частным случаем задач нелинейного программирования являются задачи квадратичного программирования, в которых ограничения линейны, а целевая функция представляет собой сумму линейной и квадратичной формы:

$min\ f(x)$

$Dx=c$

$Ax \le b$

$e_i \le x_i \le g_i,\ i=1,2,\ldots,n$

Здесь $x=(x_1,x_2,\ldots,x_n)^T$,

$$f(x) = p_1x_1 + p_2x_2 + \ldots + p_nx_n + q_{11}x_1^2 + q_{22}x_2^2 + \ldots + q_{nn}x_n^2 + q_{12}x_1x_2 +$$
$$+ q_{13}x_{13} + \ldots + q_{1n}x_{1n} + q_{23}x_{23} + \ldots + q_{2n}x_2x_n + \ldots + q_{n-1n}x_{n-1}x_n$$

$$A = \begin{pmatrix} a_{11}\ a_{12}\ \ldots\ a_{1n} \\ a_{21}\ a_{22}\ \ldots\ a_{2n} \\ \ldots\ldots\ldots\ldots\ldots \\ a_{m1}\ a_{m2}\ \ldots\ a_{mn} \end{pmatrix}, D = \begin{pmatrix} d_{11}\ d_{12}\ \ldots\ d_{1n} \\ d_{21}\ d_{22}\ \ldots\ d_{2n} \\ \ldots\ldots\ldots\ldots\ldots \\ d_{k1}\ d_{k2}\ \ldots\ d_{kn} \end{pmatrix}, b = \begin{pmatrix} b_1 \\ b_2 \\ \ldots \\ b_m \end{pmatrix}, c = \begin{pmatrix} c_1 \\ c_2 \\ \ldots \\ c_k \end{pmatrix}, p = \begin{pmatrix} p_1 \\ p_2 \\ \ldots \\ p_n \end{pmatrix}.$$

В матричной форме целевая функция может быть записана как

$$f(x) = \frac{1}{2}x^TQx + p^Tx, \text{ где } Q = \begin{pmatrix} 2q_{11}\ q_{12}\ \ldots q_{1n} \\ q_{21}\ 2q_{22}\ \ldots q_{2n} \\ \ldots\ldots\ldots\ldots\ldots \\ q_{n1}\ q_{n2}\ \ldots 2q_{nn} \end{pmatrix}, q_{ij} = q_{ji}, i, j = \overline{1,n}.$$

Для решения задачи квадратичного программирования в Scilab предназначена функция *qld*. С учетом приведенных выше обозначений ее синтаксис выглядит так:

[x [,lagr [,info]]] = qld(Q, p, E, z, ei, gi, m [,tol])

Аргументами функции *qld* являются:

Q – положительно определенная симметричная матрица размерности *n*x*n*;

p – вектор-столбец размерности *n*;

E – матрица размерности (*m*+ *k*) x *n*) (E=[D;A]);

z – вектор-столбец размерности *m*+ *k* (z=[c;b]);

ei – вектор-столбец размерности *n*. Ограничение снизу на переменную x_i ($e_i \le x_i$). Если нет ограничений снизу, можно задать ei=[]. Если некоторые компоненты вектора *x* ограничены снизу,

то можно задать значения других, неограниченных снизу компонент вектора x, равными очень маленькому числу (например, -10000);

gi – вектор-столбец верхних границ на переменные x_i (см. выше);

m – число ограничений типа равенств (то есть D(1:m,:)*x= z(1:m));

tol – требуемая точность вычислений;

x – найденное оптимальное решение;

lagr – вектор множителей Лагранжа [13]. Если значения переменных x_i ограничены снизу и сверху, то размерность вектора lagr равна $m+k+2n$. $m+k$ первых компонент вектора связаны с линейными ограничениями, остальные – с ограничениями на компоненты вектора x. Если ограничение сверху (снизу) на переменную x_i задано, то значение lagr(i)>0 (соответственно, меньше нуля). Если ограничений нет, то вектор содержит $m+k$ компонент.

В случае успешного решения задачи все значения lagr, относящиеся к ограничениям и неравенствам должны быть не меньше нуля.

Значения параметра info могут быть следующими:

info==1 : Требуется слишком большое число итераций;

info==2 : Недостаточная точность для удовлетворения условия сходимости;

info==5 : Длина обрабатываемого массива слишком мала;

info==10: Конфликтующие ограничения.

Пример 5.3. Найти минимум функции *f(x)*:

$$f(x) = x_1^2 + x_2^2 + x_3^2 - 2x_1x_2 - x_1 - x_2 + x_3$$

$$x_1 + x_2 + x_3 \geq 1$$

$$2x_1 + x_2 - x_3 \geq -1$$

$$x_1 - x_2 + x_3 = 0$$

$$0 \leq x_1 \leq 1$$

$$0 \leq x_2 \leq 1$$

$$0 \leq x_3 \leq 1$$

$$A=\begin{pmatrix}-1 & -1 & -1\\ -2 & -1 & 1\end{pmatrix}, D=\begin{pmatrix}1 & -1 & 1\end{pmatrix}, b=\begin{pmatrix}-1\\ 1\end{pmatrix}, c=\begin{pmatrix}0\end{pmatrix}, p=\begin{pmatrix}-1\\ -1\\ 1\end{pmatrix}, e=\begin{pmatrix}0\\ 0\\ 0\end{pmatrix}, g=\begin{pmatrix}1\\ 1\\ 1\end{pmatrix},$$

$$Q=\begin{pmatrix}2 & -2 & 0\\ -2 & 2 & 0\\ 0 & 0 & 2\end{pmatrix}.$$

Решение.

```
clc
D= [1,-1,1];
c=[0];
//D*x = c (1 ограничение-равенство, то есть m=1)
A=[-1,-1,-1;
   -2,-1,1];
b=[-1;1];
//A*x <= b (2 ограничения типа неравенств, то есть k=2)
//  x между ei и gi:
ei=[0;0;0];
gi=[1;1;1];
//минимизтруем 0.5*x'*Q*x + p'*x
p=[-1;-1;1];
Q=[2 -2 0;-2 2 0;0 0 2];
E=[D;A];
z=[c;b];
m=1;
[x,lagr]=qld(Q,p,E,z,ei,gi,m)
disp(x,'x=')
disp(0.5*x'*Q*x + p'*x,'Значение целевой функции равно ')
```

```
x=

   1.

   1.

  -2.465D-32

 Значение целевой функции равно   -2.
```

Пример 5.4. Найти минимум функции:

$$f(x)=\frac{1}{2}x_1^2+x_2^2-x_1x_2-2x_1-6x_2$$

$$x_1+x_2\le 2$$
$$-x_1+2x_2\le 2$$
$$2x_1+x_2\le 3$$
$$0\le x_1, 0\le x_2$$

Решение.

```
clc
clear
D= [];
c=[];
//D*x = c (нет ограничений типа равенств, то есть m=0)
A=[1 1;-1 2;2 1];
b=[2;2;3];
//A*x <= b (3 ограничения типа неравенства k=3)
// x между ei и gi:
ei=[0;0];gi=[];//нет ограничений сверху
//минимизируется функция 0.5*x'*Q*x + p'*x
p=[-2;-6]; Q=[1 -1;-1 2];
E=[D;A];
z=[c;b];
m=0;
[x,lagr]=qld(Q,p,E,z,ei,gi,m)
disp(x,'x=')
disp(0.5*x'*Q*x + p'*x, 'Значение целевой функции равно ')
```

Оптимальный план:

x=

0.6666667

1.3333333

Значение целевой функции равно -8.2222222

Попытка решить задачу поиска минимума функции

$$f(x) = x_1^2 + x_2^2$$
$$x_1 + x_2 = 2$$
$$x_1 + 2x_2 \geq 10$$
$$0 \leq x_1, 0 \leq x_2$$

привело к выдаче сообщения:

qld: Граничные условия противоречивы.

Это означает, что множество решений (планов) задачи пусто.

Решение задачи

$$\min\ f(x) = x_1^2 - x_1 - 4x_2,$$
$$x_1 + x_2 \geq 2$$
$$0 \leq x_1, 0 \leq x_2$$

дает:

x=

0.5

3.603D+16

Значение целевой функции равно -1.441D+17

Целевая функция задачи не ограничена снизу.

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

Задание 1. Найти минимум функции $y=\alpha x^4-(10\mu+\nu)x^3-(\gamma+\theta)x+34$.

Задание 2. Найти минимум функции

$$y = x_1^4 - 2\mu x_1 x_2 + (\nu+1)x_2^2 + x_3^3 - 3x_1 x_3 - 2x_2 + \alpha.$$

Задание 3. Решить задачу линейного программирования

$$\begin{aligned} &\min -x_1 - 2x_2 + (\mu+1)x_3 \\ &x_1 + x_2 + x_3 = 30 \\ &-\alpha x_1 - 6x_2 \geq -60 - \nu \\ &\mu x_1 + \nu x_2 + x_3 \leq 82 \\ &x \geq 0 \end{aligned}$$

Задание 4. Решить задачу квадратичного программирования

$$\begin{aligned} &\min f(x) = \mu x_1^2 + \nu x_2^2 + 2x_3^2 - \gamma x_1 x_2 + \beta x_1 x_3 - \theta x_1 + 2x_2 + 3x_3 \\ &x_1 + x_2 - x_3 \geq 100 \\ &\nu x_1 + x_2 - \alpha x_3 \leq 10\mu + \nu + 59 \\ &2x_1 - x_2 + x_3 = \alpha + 4 \\ &0 \leq x_1,\ 0 \leq x_2,\ 0 \leq x_3 \end{aligned}$$

Задание 5. Решить задачу квадратичного программирования

$$\begin{aligned} &\max f(x) = \mu x_1^2 - \nu x_2^2 - \gamma x_1 x_2 - \theta x_1 \\ &x_1 + x_2 \geq 10 \\ &\nu x_1 + x_2 \leq \nu + 59 \\ &0 \leq x_1,\ 0 \leq x_2,\ 0 \leq x_3 \end{aligned}$$

ЛИТЕРАТУРА

1. Шарый, С.П. Курс вычислительных методов / С.П. Шарый. – Институт вычислительных технологий СО РАН, Новосибирский государственный университет, 2020. – 639 с.

2. Самарский, А.А. Численные методы / А.А. Самарский, А.В. Гулин. – М.: Наука, 1989. – 432 с.

3. Демидович, Б.П. Основы вычислительной математики / Б.П. Демидович, И.А. Марон. – М.: Наука, 1970. – 664 с.

4. Мудров, В. И. Метод наименьших модулей / В. И. Мудров, В. Л. Кушко. – М.: Знание, 1971. – 64 с.

5. Брановицкая, С.В. Вычислительная математика в химии и химической технологии / С.В. Брановицкая. – Киев, Издательское объединение «Вища школа», 1986. – 216 с.

6. Scilab: Решение инженерных и математических задач / Е.Р. Алексеев [и др.]. – М.: БИНОМ, 2008. – 260 с.

7. Решение инженерных задач в Scilab: учебное пособие / А.Б. Андриевский [и др.]. – СПб.: НИУ ИТМО, 2013. – 97 с.

8. Ерин, С.В. Scilab – примеры и задачи: учебное пособие / С.В. Ерин. – М.: Лаборатория «Знания будущего», 2017. – 154 с.

9. Michael Baudin. Введение в Scilab. Перевод Artem Glebov, 2010. – 85 с. [Электронный ресурс]. Свободный.

10. Кобзарь, А.И. Прикладная математическая статистика: для инженеров и научных работников / А.И. Кобзарь. – М.: Физматлит, 2006. – 816 с.

11. Орлов, А.И. Прикладная статистика / А.И. Орлов. – М.: Экзамен, 2004. – 656 с.

12. Титов, А.Н. Решение задач теории вероятностей и математической статистики в среде Scilab: учебно-методическое пособие / А.Н. Титов, Р.Ф. Тазиева. – Казань: Изд-во КНИТУ, 2019. – 120 с.

13. Лутманов, С. В. Курс лекций по методам оптимизации. – Ижевск: НИЦ РХД, 2001. – 368 с.

14. Гольдштейн, А. Л. Оптимизация в среде Matlab: учеб. пособие / А. Л. Гольдштейн. – Пермь: Изд-во Перм. нац. исслед. политехн. ун-та, 2015. – 192 с.

15. Крутова, В.И. Основы научных исследований: учебник для вузов / В.И. Крутова, В.В. Попова. – М.: Высшая школа, 1989. – 400 с.

16. Титов, А.Н. Решение задач линейной алгебры и прикладной математики в среде Scilab: учебно-методическое пособие / А.Н. Титов, Р.Ф. Тазиева. – Казан. нац. исслед. технол. ун-т. – Казань: Изд-во КНИТУ, 2020. – 98 с.