Московский Авиационный Институт

(Государственный Технический Университет)

Факультет прикладной математики и физики.

Кафедра вычислительной математики и программирования.

**Лабораторная работа №3**

**по курсу «Численные методы»**

VI семестр

Студент Баскаков О.А.

Группа 08-306

Вариант 1

Москва, 2011.

# Постановка задачи

## 1 Интерполяция многочленом

Используя таблицу значений функции , вычисленных в точках , построить интерполяционные многочлены Лагранжа и Ньютона, проходящие через точки }. Вычислить значение погрешности интерполяции в точке .

## 2 Интерполяция сплайнами

Построить кубический сплайн для функции, заданной в узлах интерполяции, предполагая, что сплайн имеет нулевую кривизну при и . Вычислить значение функции в точке .

## 3 Метод наименьших квадратов

Для таблично заданной функции путем решения нормальной системы МНК найти приближающие многочлены a) 1-ой и б) 2-ой степени. Для каждого из приближающих многочленов вычислить сумму квадратов ошибок. Построить графики приближаемой функции и приближающих многочленов.

## 4 Численное дифференцирование

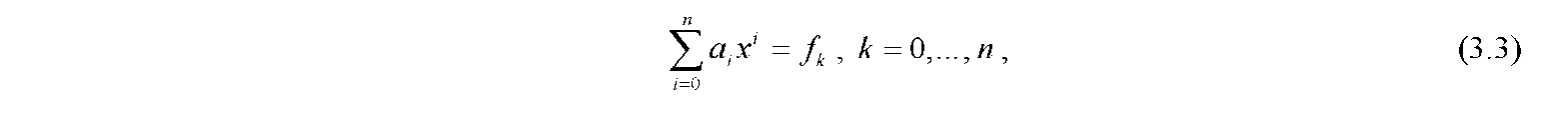
Вычислить первую и вторую производную от таблично заданной функции в точке .

## 5 Численное интегрирование

Вычислить определенный интеграл методами прямоугольников, трапеций, Симпсона с шагами . Оценить погрешность вычислений, используя Метод Рунге-Ромберга.

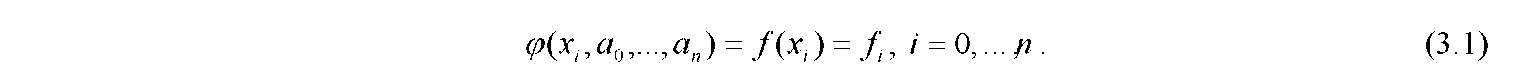
# Теоретическая часть ИНТЕРПОЛЯЦИЯ

Пусть на отрезке [a, b] задано множество несовпадающих точек xi



(интерполяционных узлов), в которых известны значения функции f = f (xi), i = 0,... n .

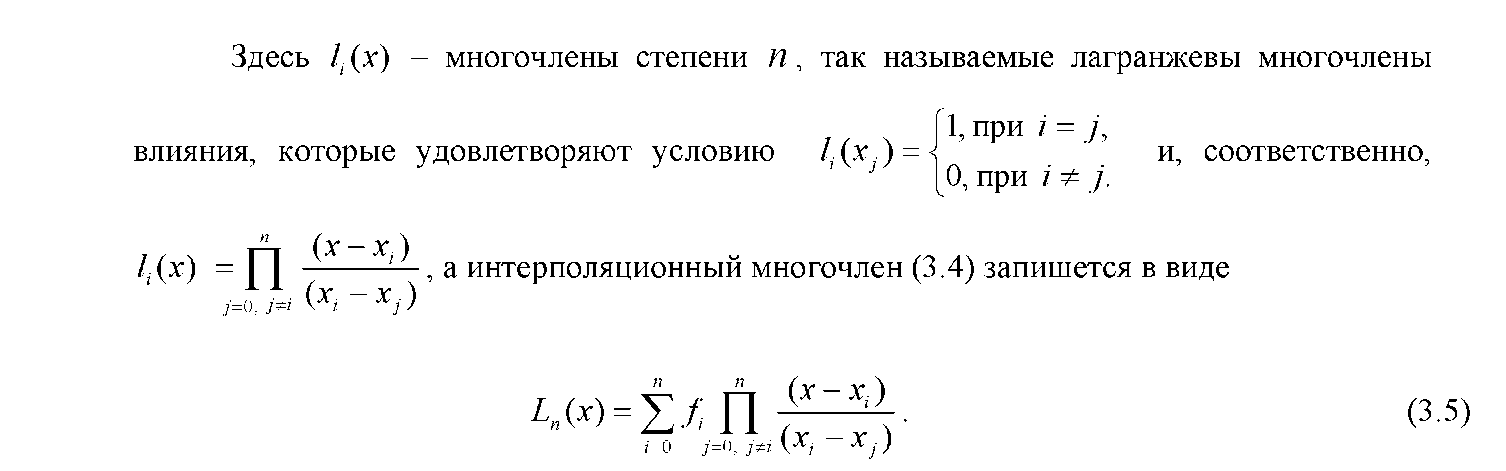
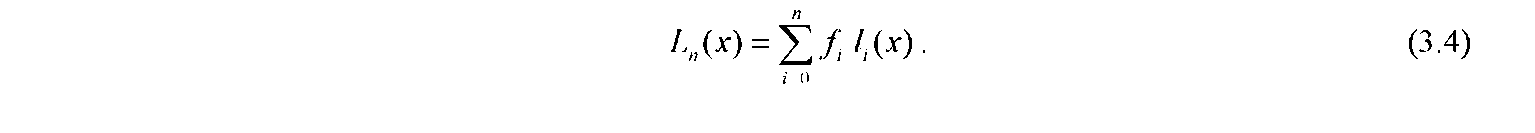
Приближающая функция p(x, a) такая, что выполняются равенства



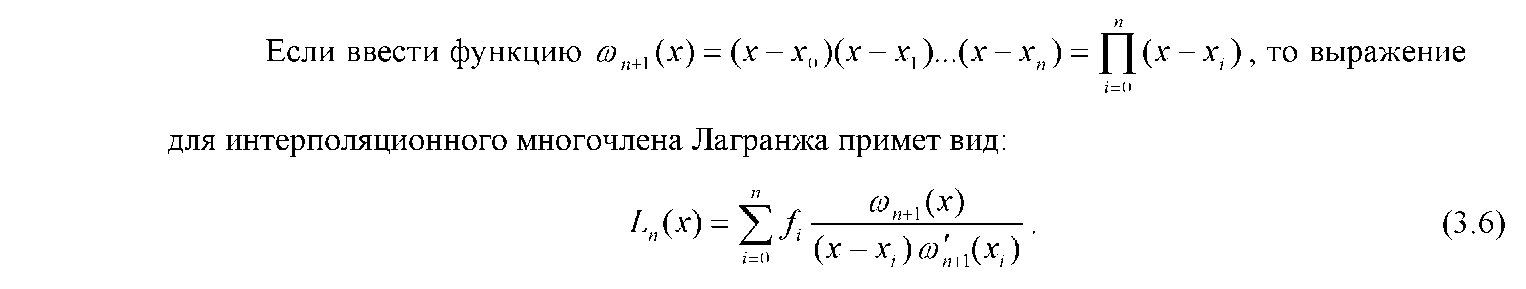
называется интерполяционной.

Наиболее часто в качестве приближающей функции используют многочлены степени n.

Для нахождения интерполяционного многочлена не обязательно решать систему (3.3). Произвольный многочлен может быть записан в виде:



Интерполяционный многочлен, записанный в форме (3.5), называется ин­терполяционным многочленом Лагранжа.

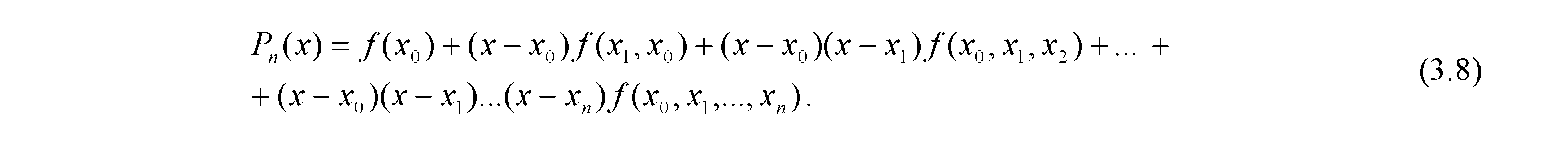


Недостатком интерполяционного многочлена Лагранжа является необходимость полного пересчета всех коэффициентов в случае добавления дополнительных интерполяционных узлов. Чтобы избежать указанного недостатка используют интерполяционный многочлен в форме Ньютона.

Введем понятие разделенной разности. Разделенные разности нулевого порядка совпадают со значениями функции в узлах. Разделенные разности первого порядка обозначаются f (xt, x}.) и определяются через разделенные разности нулевого порядка.

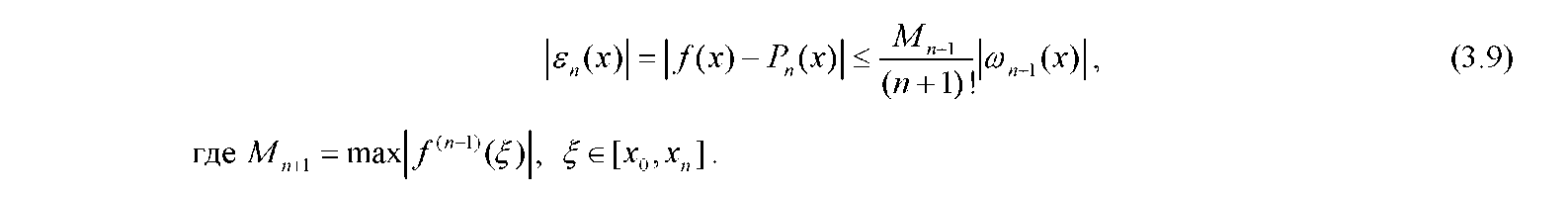
Пусть известны значения аппроксимируемой функции f (x) в точках x0,x1,...,xn.

Интерполяционный многочлен, значения которого в узлах интерполяции совпадают со значениями функции f( x) может быть записан в виде:



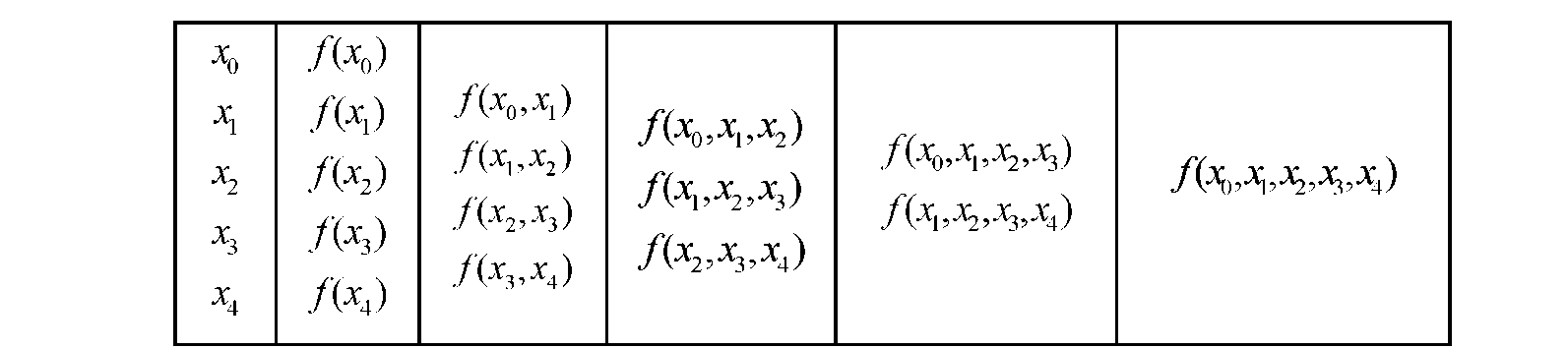
Запись многочлена в формуле (3.8) есть так называемый интерполяционный многочлен Ньютона. Если функция f (x) не есть многочлен n-й степени, то формула (3.8) для Pn(x) приближает функцию f (x) с некоторой погрешностью. Отметим, что при добавлении новых узлов первые члены многочлена Ньютона остаются неизменными.

Если функция задана в точках x0 , x1,... , xn , то при построении интерполяционного



многочлена Ньютона удобно пользоваться таблицей, называемой таблицей разделенных разностей, пример которой для n = 4 приведен в табл. 3.1.

Таблица 3.1

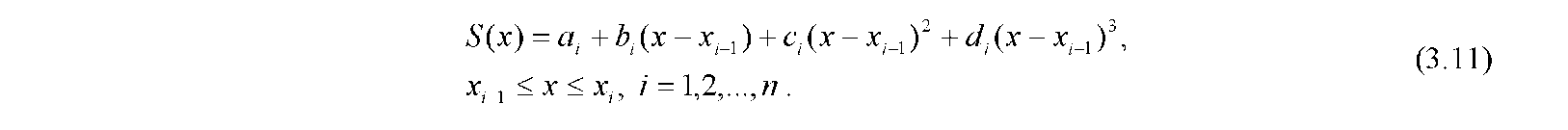


Для повышения точности интерполяции в сумму (3.8) могут быть добавлены новые члены, что требует подключения дополнительных интерполяционных узлов. При этом безразлично, в каком порядке подключаются новые узлы. Этим формула Ньютона выгодно отличается от формулы Лагранжа.

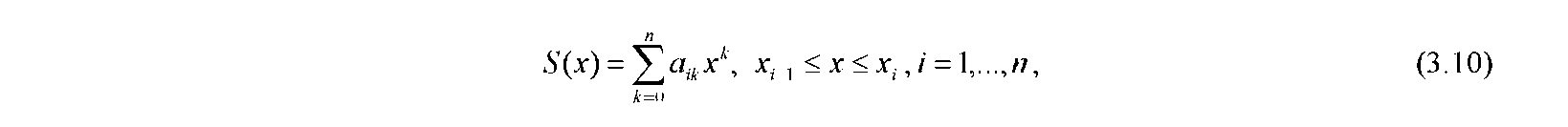
Погрешность интерполяционных многочленов Лагранжа и Ньютона для случая аналитически заданной функции f(x) априорно может быть оценена по формуле, вывод которой приводится, например в [1].

Если величину производных аппроксимируемой функции оценить сложно (например, для таблично заданной функции), то используется апостериорная оценка по первому отброшенному члену интерполяционного многочлена Ньютона, в который входят разделенные разности, являющиеся аналогами производных соответствующих порядков

Использование одной интерполяционной формулы на большом числе узлов нецелесообразно. Интерполяционный многочлен может проявить свои колебательные свойства, его значения между узлами могут сильно отличаться от значений интерполируемой функции. Одна из возможностей преодоления этого недостатка заключается в применении сплайн-интерполяции. Суть сплайн-интерполяции заключается в определении интерполирующей функции по формулам одного типа для различных непересекающихся промежутков и в стыковке значений функции и её производных на их границах.

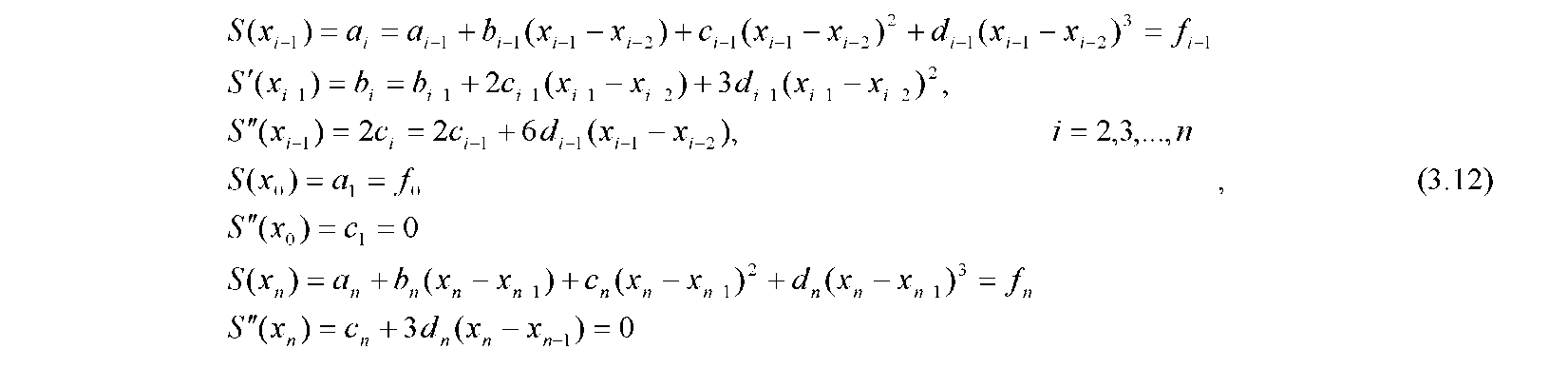


Наиболее широко применяемым является случай, когда между любыми двумя точками разбиения исходного отрезка строится многочлен n-й степени:

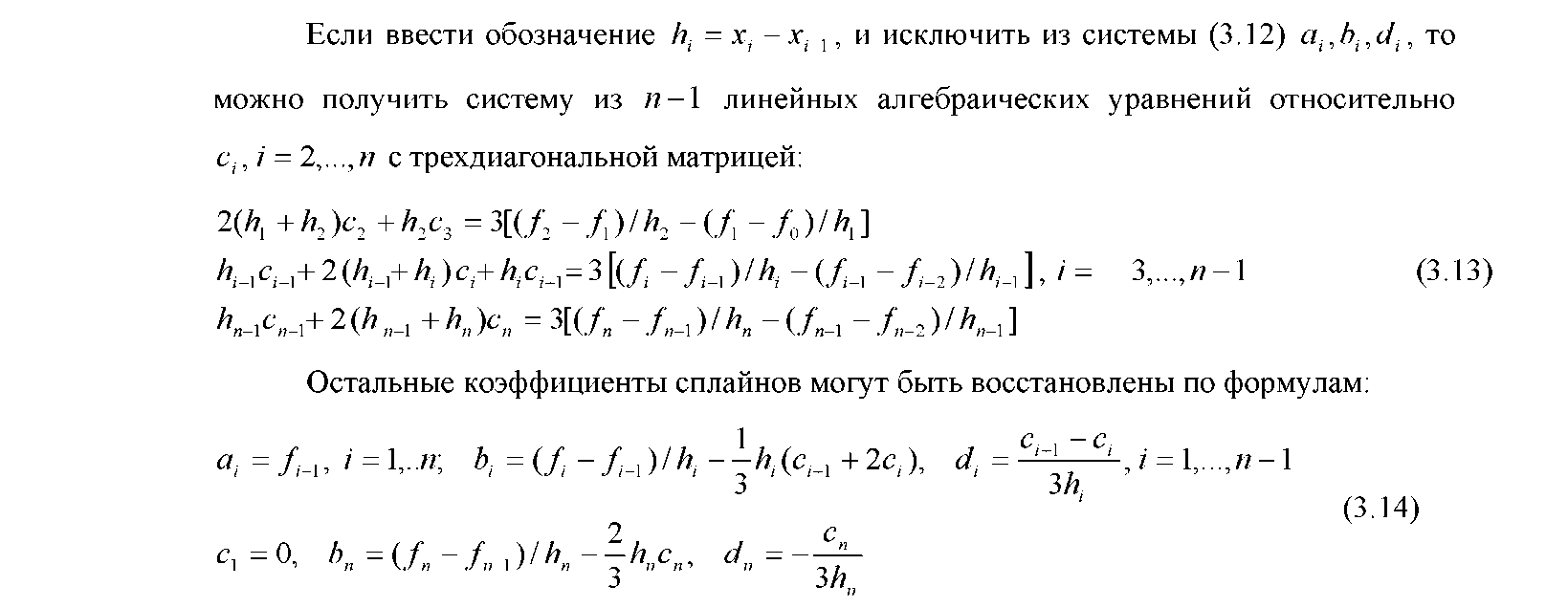


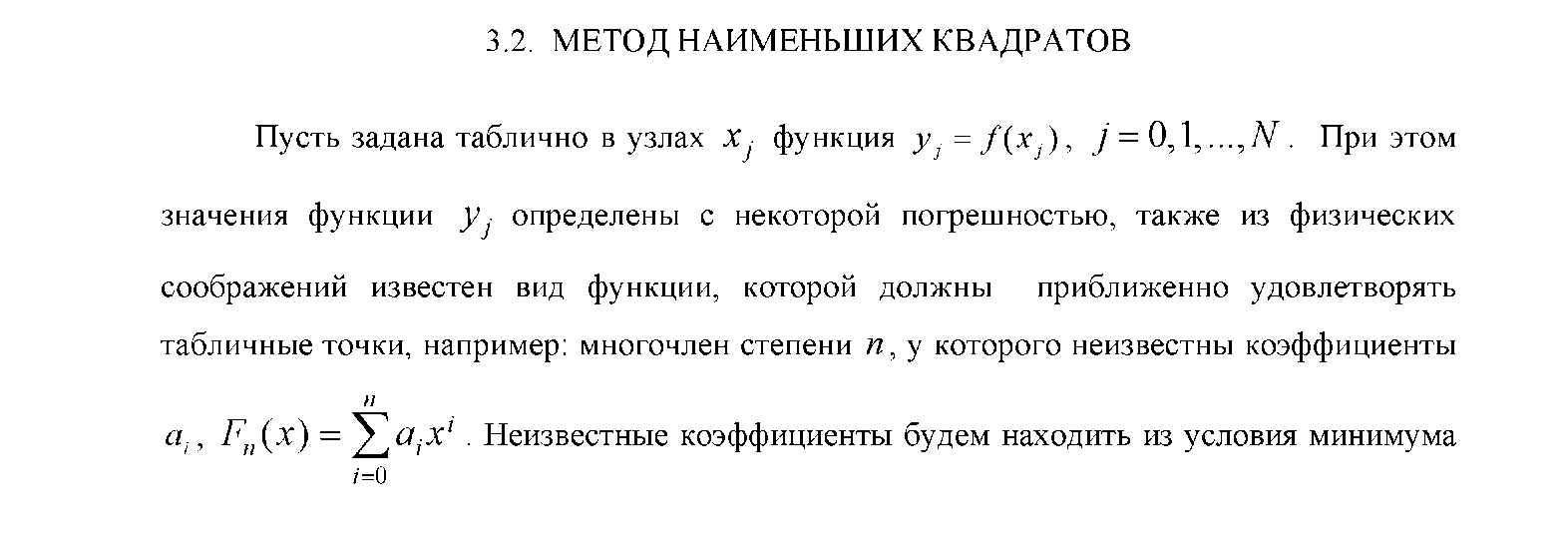
который в узлах интерполяции принимает значения аппрокимируемой функции и непрерывен вместе со своими (n -1) производными. Такой кусочно-непрерывный интерполяционный многочлен называется сплайном. Его коэффициенты находятся из условий равенства в узлах сетки значений сплайна и приближаемой функции, а также равенства n -1 производных соответствующих многочленов. На практике наиболее часто используется интерполяционный многочлен третьей степени, который удобно представить как

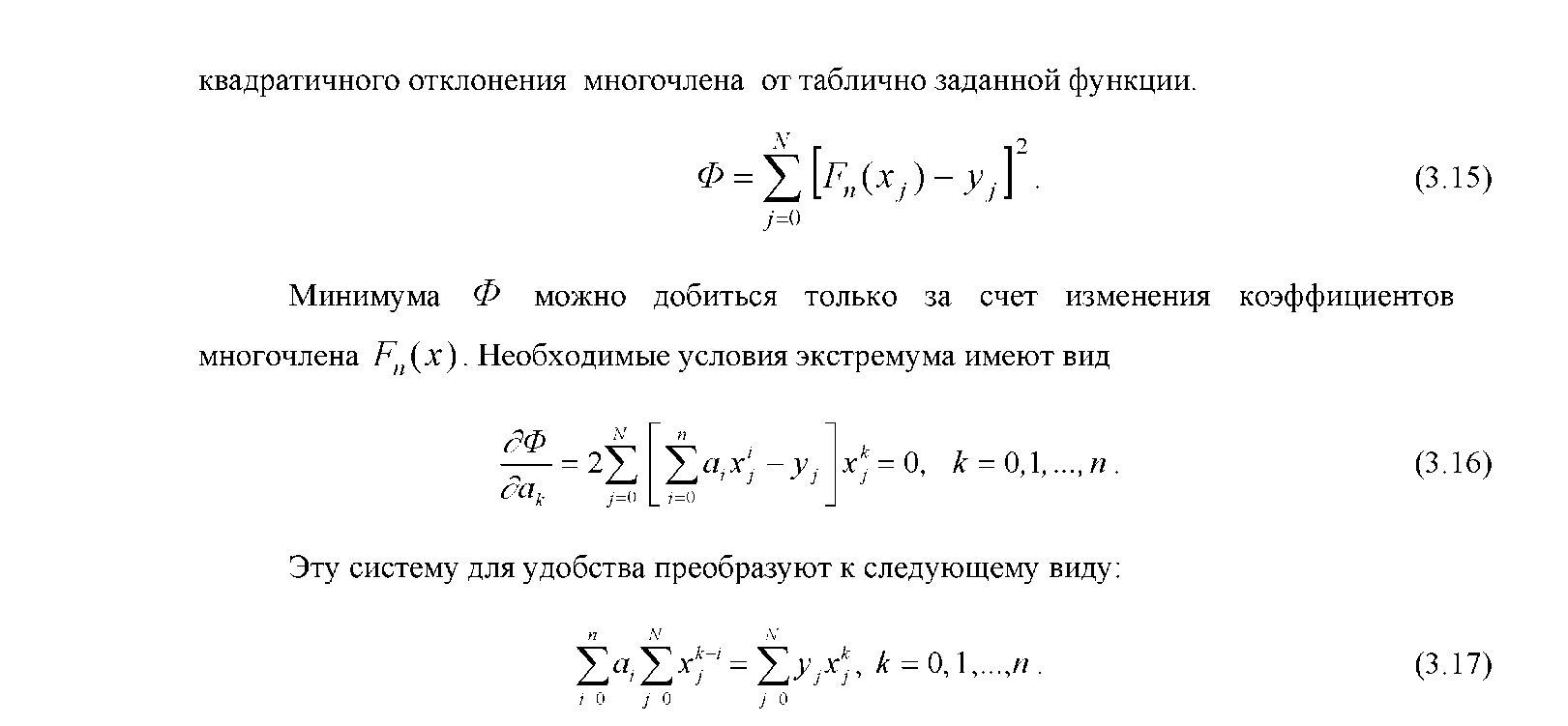
Для построения кубического сплайна необходимо построить n многочленов третьей степени, т.е. определить 4n неизвестных ai, bi, ci, di. Эти коэффициенты ищутся из условий в узлах сетки.



В (3.12) предполагается, что сплайны имеют нулевую кривизну на концах отрезка. В общем случае могут быть использованы и другие условия.

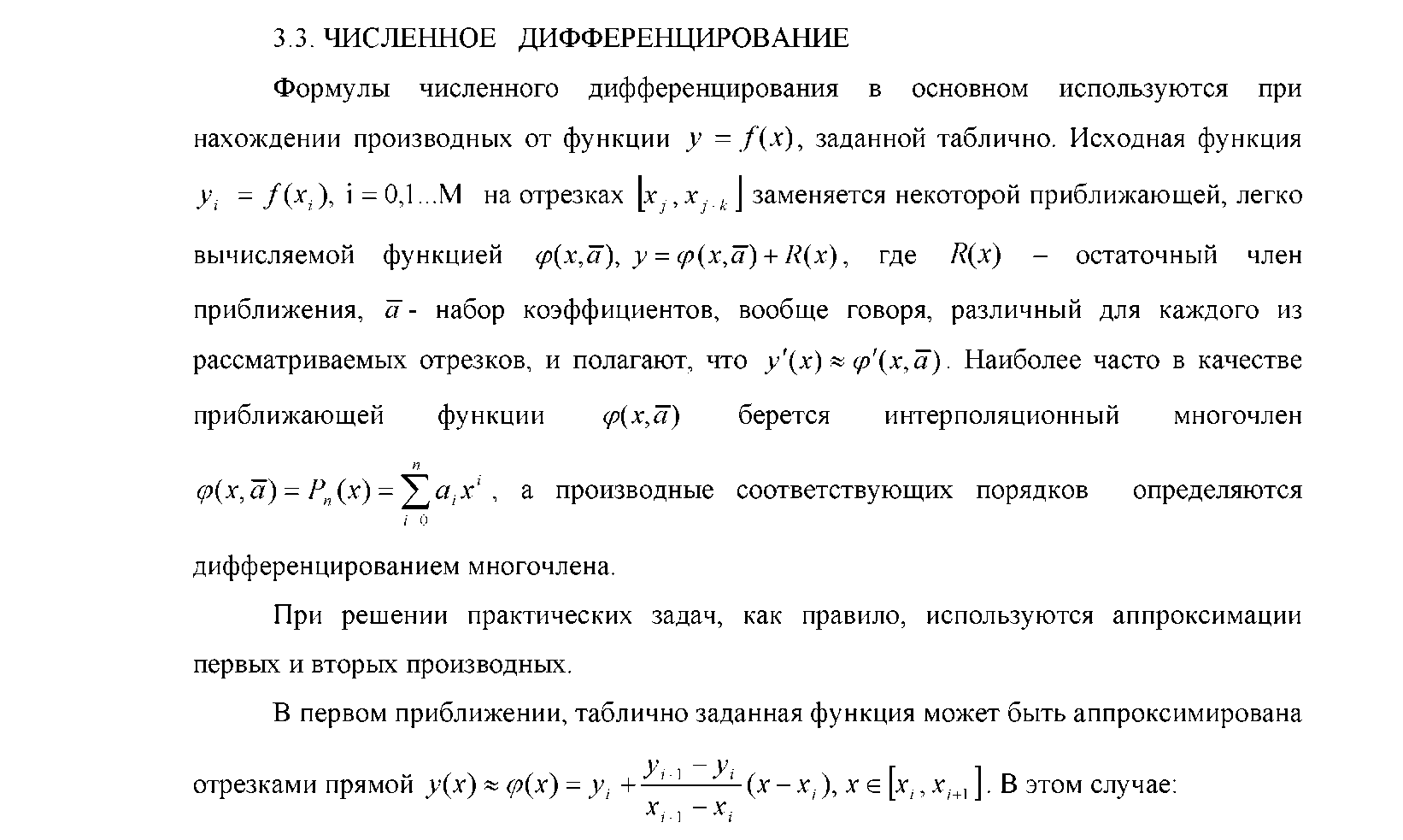


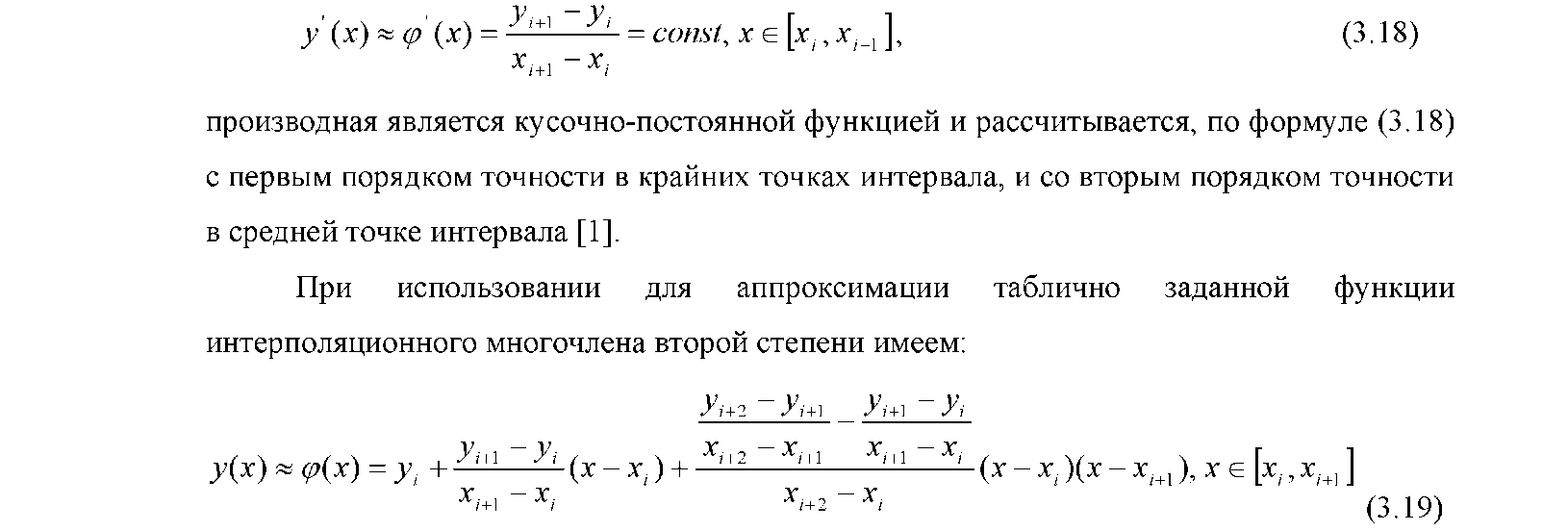


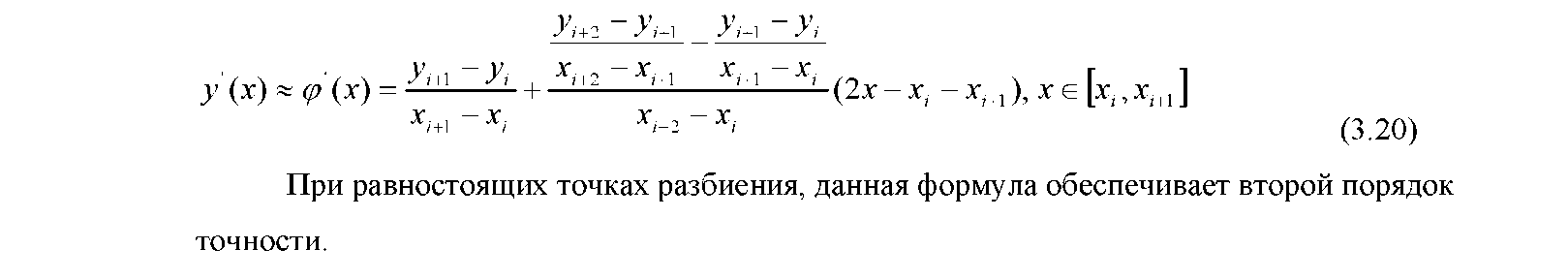


Система (3.17) называется нормальной системой метода наименьших квадратов (МНК) представляет собой систему линейных алгебраических уравнений относительно коэффициентов а.. Решив систему, построим многочлен Fn (X), приближающий функцию / ( x) и минимизирующий квадратичное отклонение.

Необходимо отметить, что система (3.17) с увеличением степени n приближающего многочлена становится плохо обусловленной и решение её связано с большой потерей точности. Поэтому при использовании метода наименьших квадратов, как правило, используют приближающий многочлен не выше третьей степени.

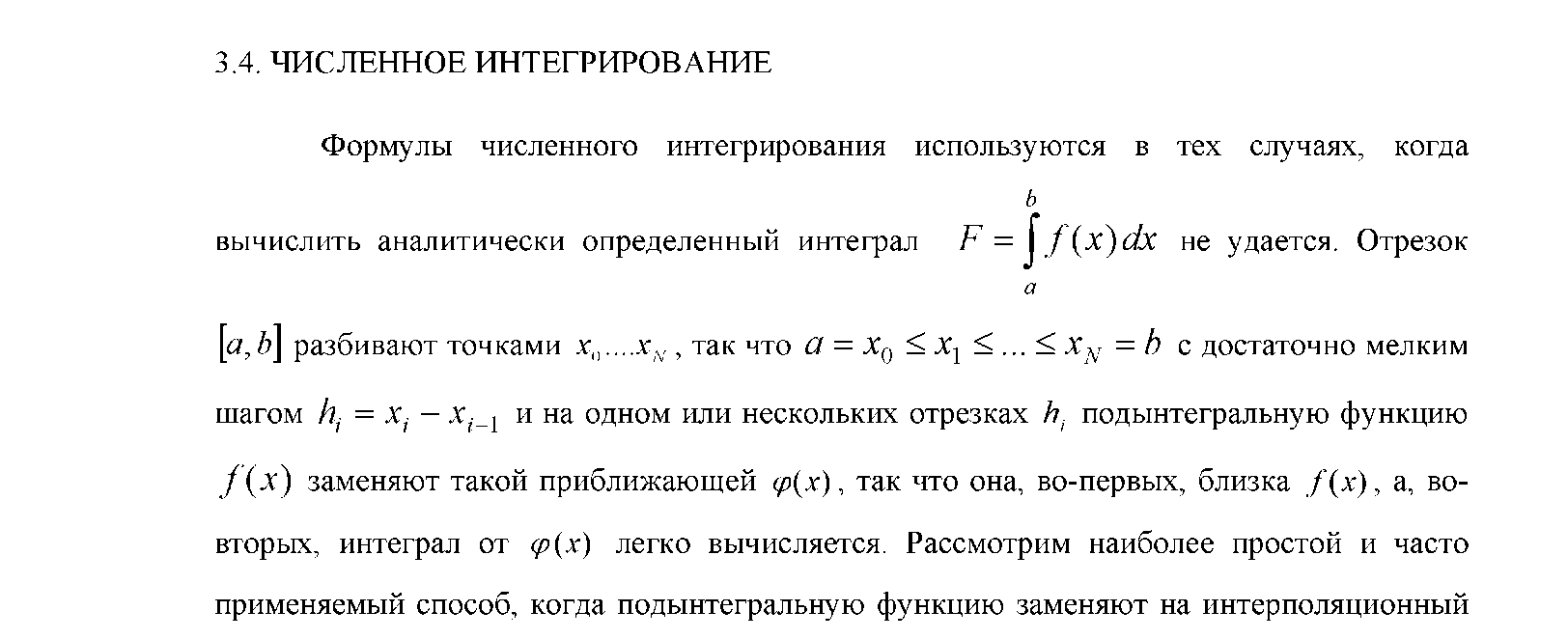


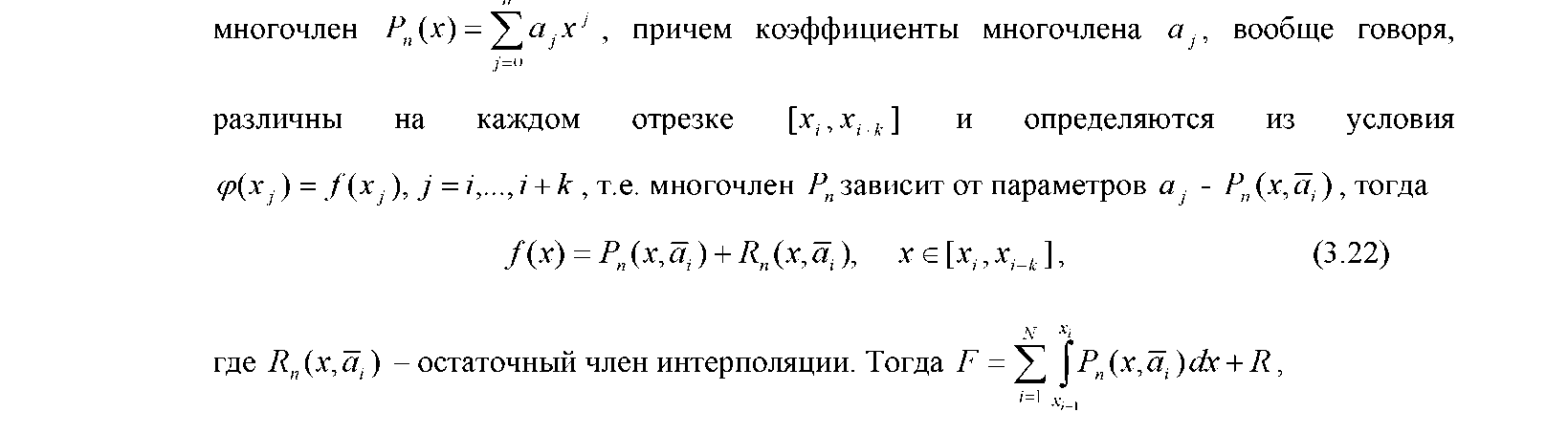


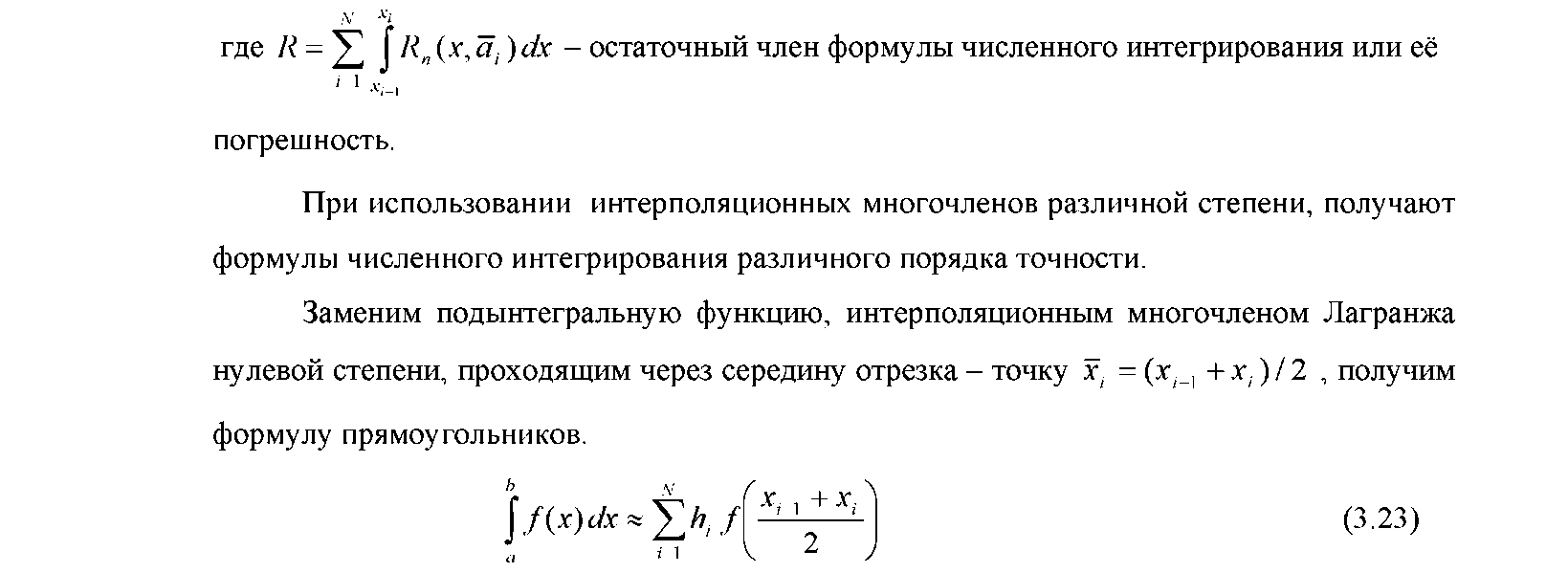


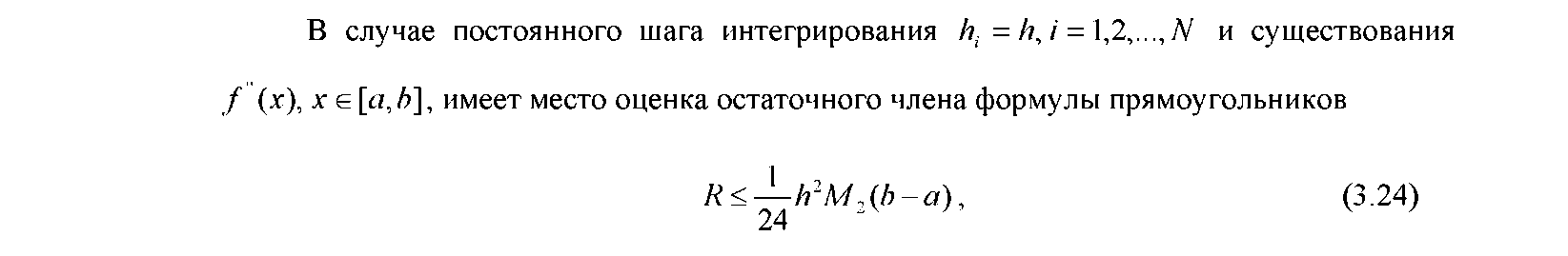
Для вычисления второй производной, необходимо использовать

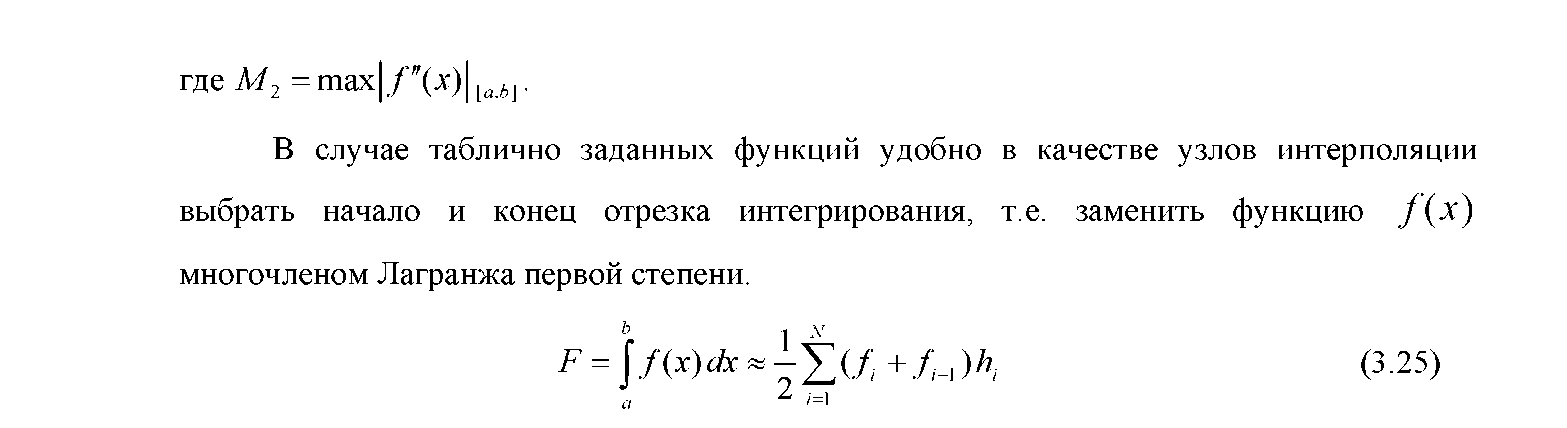
интерполяционный многочлен, как минимум второй степени. После дифференцирования многочлена получаем

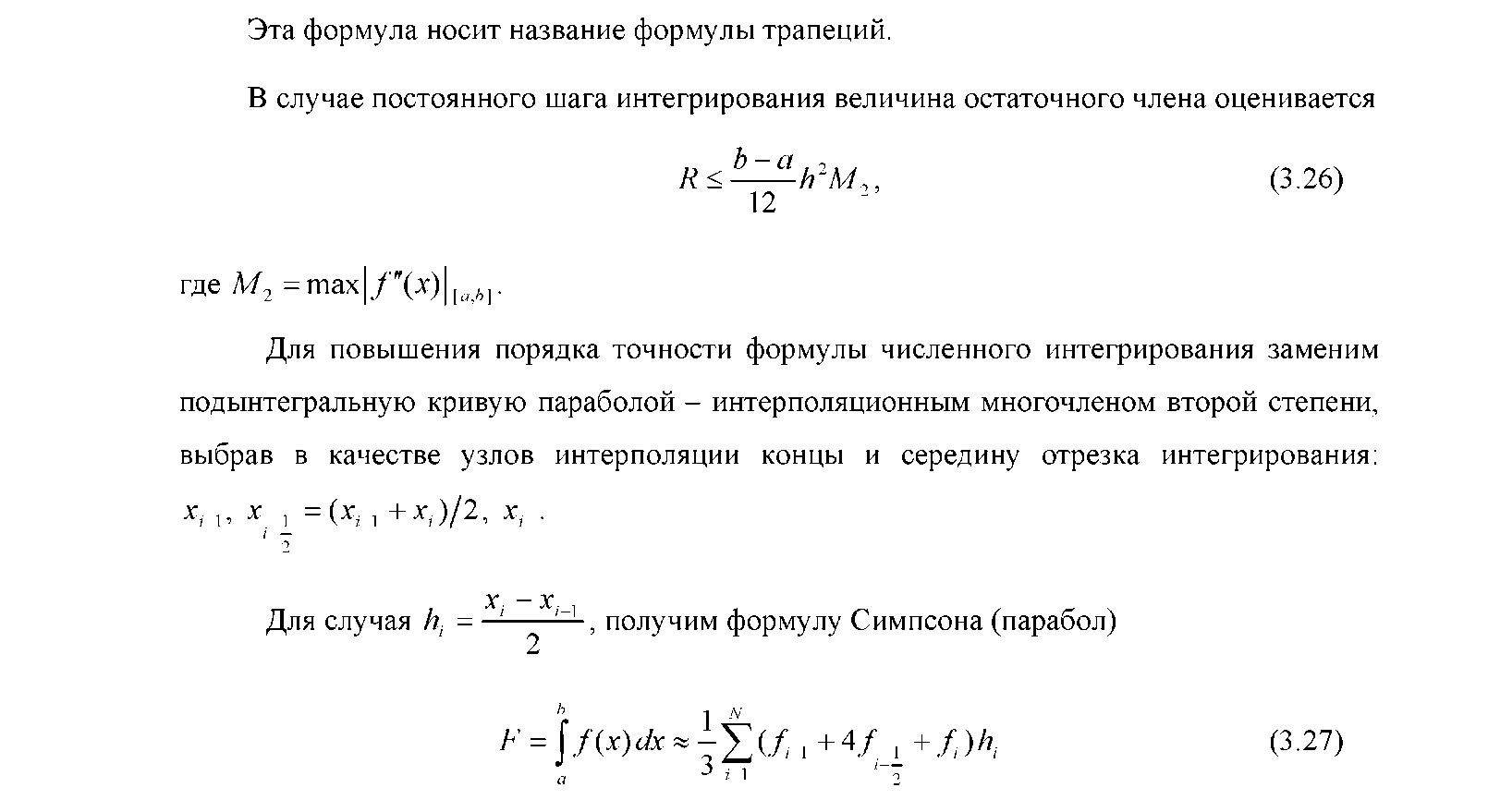


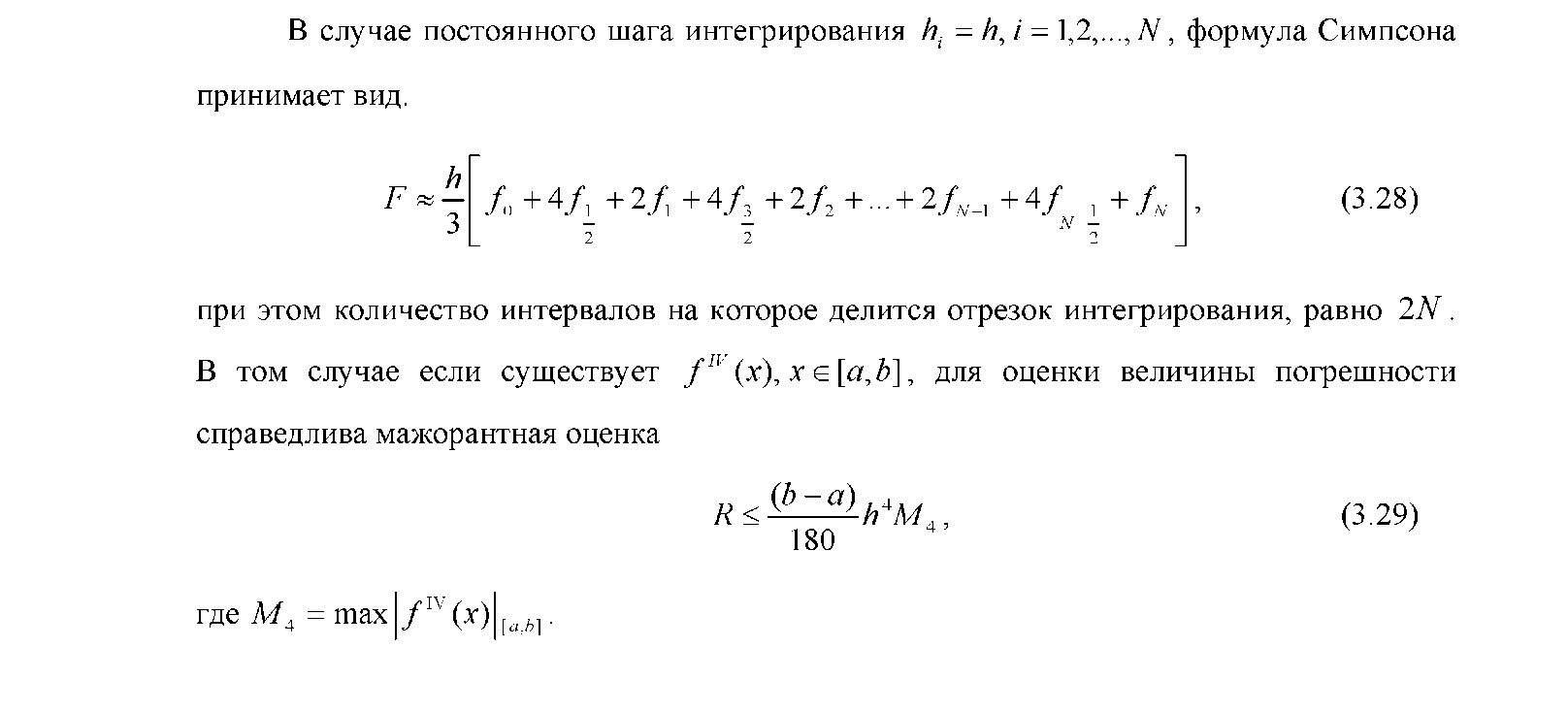












# Исходный код на языке Python

## 1 Интерполяция многочленом

#! /usr/bin/python3.1

# -\*- coding: utf-8 -\*-

from math import sqrt,sin,cos,atan,pi,log

from copy import copy, deepcopy

from functools import reduce

Xi = [0.1\*pi, 0.2\*pi, 0.3\*pi, 0.4\*pi]

Yi = [sin(xx) for xx in Xi]

Y1\_global = []

#Xi = [0.0, 1.0, 2.0, 3.0]

#Yi = [round(sin(xx\*pi/6), 5) for xx in Xi]

X0 = pi/4

Y0 = sin(X0)

def product(L):

mul = lambda x, y: x\*y

return reduce(mul, L)

def l\_i(x, i, X):

""" OK """

n = len(X)

S = 1

for j in range(n):

if (i!=j):

S \*= (x - X[j])/(X[i] - X[j])

return S

def L\_n(x, Y, X):

return sum( [(Y[i]\*l\_i(x, i, X)) for i in range(len(Y))] )

def L(Y, X):

return lambda x: L\_n(x, Y, X)

def Y\_m(Y, X):

m = len(Y)

if (m == 1):

return Y[0]

elif (m == 2):

return(Y[0] - Y[1])/(X[0] - X[1])

else:

return(Y\_m(Y[0:m-1],X[0:m-1])-Y\_m(Y[1:m],X[1:m]) )/(X[0]-X[m-1])

def P\_m(X):

if (len(X) == 0): return lambda x: 1

return lambda x:product( [(x - x\_i) for x\_i in X])

def N\_1(x, Y1, P1):

return sum( [(Y\*P(x)) for (Y,P) in zip(Y1,P1)] )

def N(Y, X):

m = len(X)

Y1 = [Y\_m(Y[0:i],X[0:i]) for i in range(1,m+1)]

P1 = [P\_m(X[0:i]) for i in range(0,m)]

print("polynom koef:")

for yy in Y1: print(round(yy, 5))

return lambda x: N\_1(x,Y1,P1)

def main():

print("table function: ")

print(Xi)

print(Yi)

lagranje = L(Yi, Xi)

newton = N(Yi, Xi)

print("Test lagranje", Yi[2], "=", lagranje(Xi[2]) )

print("Test newton ", Yi[2], "=", newton (Xi[2]) )

eps\_n = abs( Y0 - lagranje(X0) )

print( "|sin(x) - P\_n(x)| = ", eps\_n)

return 0

main()

## 2 Интерполяция сплайнами

#! /usr/bin/python3.1

# -\*- coding: utf-8 -\*-

from math import sqrt,sin,cos,atan,pi,log

from copy import copy, deepcopy

from functools import reduce

class Tridiagonal\_Matrix:

def \_\_init\_\_(self):

self.a = []

self.b = []

self.c = []

self.d = []

self.n = 0

def solve(self):

"""Method progonki"""

a = self.a

b = self.b

c = self.c

d = self.d

n = len(d)

P = []

Q = []

P.append(-c[0]/b[0])

Q.append( d[0]/b[0])

for i in range(1, n):

P.append( -c[i] / (b[i]+a[i]\*P[i-1]) )

Q.append( (d[i] - a[i]\*Q[i-1]) / (b[i] + a[i]\*P[i-1]) )

x = [0]\*n

x[n-1] = Q[n-1]

for i in range(n-2, -1, -1):

x[i] = P[i]\*x[i+1] + Q[i]

return x

Xi = [0.0, 1.0, 2.0, 3.0, 4.0]

#Yi = [0.0, 0.5, 0.86603, 1.0, 0.86603]

Yi = [0.0, 1.8415, 2.9093, 3.1411, 3.2432]

Hi = [0.0] + [(Xi[i] - Xi[i-1]) for i in range(1,len(Xi))]

X0 = 1.5

Y0 = 2.4969

def roundx(v):

return [round(xx, 5) for xx in v]

def build\_ci(Y, X, H):

n = len(X)

M = Tridiagonal\_Matrix()

for i in range(2, n): # numerating from zero

a = H[i-1]

b = 2\*(H[i-1]+H[i])

c = H[i]

d = 3\*( (Y[i]-Y[i-1])/H[i] - (Y[i-1]-Y[i-2])/H[i-1] )

if (i==2) : a = 0

if (i==n-1): c = 0

M.a.append(a)

M.b.append(b)

M.c.append(c)

M.d.append(d)

print("Tridiagonal\_Matrix:")

print(roundx(M.a))

print(roundx(M.b))

print(roundx(M.c))

print(roundx(M.d))

x = [0.0] + M.solve()

print ("x = ", roundx(x) )

return x

def spline1(A, B, C, D, X, x):

""" calculate spline"""

if (x < X[0]) : return 0.0

if (x > X[-1]) : return 0.0

i = 0

while (X[i+1] < x): i+=1

# segment X[i]..X[i+1]

x1 = X[i]

return A[i] + B[i]\*(x-x1) + C[i]\*(x-x1)\*\*2 + D[i]\*(x-x1)\*\*3

def cr\_spline(A, B, C, D, X):

""" create spline"""

return lambda x: spline1(A, B, C, D, X, x)

def main():

print("table function: ")

print(Xi)

print(Yi)

Ai = Yi[0 : -1]

Ci = build\_ci(Yi, Xi, Hi)

n = len(Ci)-1

Bi = [( (Yi[i+1]-Yi[i])/Hi[i+1] - Hi[i+1]\*(Ci[i+1]+2\*Ci[i])/3 ) for i in range(0,n)]

Bi.append( (Yi[n+1]-Yi[n])/Hi[n] - Hi[n]\*Ci[n]\*2/3 )

Di = [( (Ci[i+1]-Ci[i])/(3\*Hi[i+1]) ) for i in range(0,n)]

Di.append( -Ci[n]/(3\*Hi[n]) )

print("proverka:")

print("A = ", roundx(Ai))

print("B = ", roundx(Bi))

print("C = ", roundx(Ci))

print("D = ", roundx(Di))

sp = cr\_spline(Ai, Bi, Ci, Di, Xi)

print("\ntest1 ", Y0,"=", sp(X0) )

return 0

main()

## 3 Метод наименьших квадратов

#! /usr/bin/python3.1

# -\*- coding: utf-8 -\*-

from math import sqrt,sin,cos,atan,pi,log

from copy import copy, deepcopy

from functools import reduce

from matrix\_2 import \*

#Xi = [-1.0, 0.0, 1.0, 2.0 , 3.0, 4.0 ]

#Yi = [-0.5, 0.0, 0.5, 0.86603, 1.0, 0.86603]

Xi = [0.0, 1.7 , 3.4 , 5.1 , 6.8 , 8.5 ]

Yi = [0.0, 1.3038, 1.8439, 2.2583, 2.6077, 2.9155]

X0 = 1.5

Y0 = 2.4969

def roundx(v):

return [round(xx, 4) for xx in v]

def P\_n(A):

return lambda x: sum([ A[i]\*x\*\*i for i in range(0,len(A)) ] )

def cr\_MNK(Y, X, m = 2):

n = m + 1

A = Matrix("", n, n)

A.b = [sum(Y)] + [ sum([(y\_j \* x\_j\*\*k) for (y\_j, x\_j) in zip(Y, X)])

for k in range(1,n) ]

A.M = [ [sum([ x\_j\*\*(i+k) for x\_j in X ])

for k in range(n) ]

for i in range(n) ]

# A.M[0][0] = len(X) #~ does not matter

A.pr()

print("b = ", roundx(A.b) )

A.n = len(A.M)

A.m = len(A.M)

A.build\_LU()

print("p = ", A.p)

Ai = A.solve(A.shift\_b(A.b))

print("Ai = ", roundx(Ai) )

return Ai

def F\_eps(f, Y, X):

return sum( [(f(xx)-yy)\*\*2 for (xx,yy) in zip(X, Y)] )

def main():

print("table function: ")

print(Xi)

print(Yi)

Ai = cr\_MNK(Yi, Xi, 2)

polynom = P\_n(Ai)

print("\ntest1 ", Yi[1],"=", polynom (Xi[0]) )

print("F\_bolshoe =", F\_eps(polynom, Yi, Xi) )

return 0

main()

## 4 Численное дифференцирование

#! /usr/bin/python3.1

# -\*- coding: utf-8 -\*-

from math import sqrt,sin,cos,atan,pi,log

from copy import copy, deepcopy

from functools import reduce

from matrix\_2 import \*

#Xi = [-1.0, 0.0, 1.0, 2.0 , 3.0, 4.0 ]

#Yi = [-0.5, 0.0, 0.5, 0.86603, 1.0, 0.86603]

Xi = [0.0, 0.1 , 0.2 , 0.3 , 0.4 ]

Yi = [1.0, 1.1052, 1.2214, 1.3499, 1.4918]

X0 = 0.2

Y0 = 2.4969

def roundx(v):

return [round(xx, 4) for xx in v]

def P\_n(A):

return lambda x: sum([ A[i]\*x\*\*i for i in range(0,len(A)) ] )

def Differetiation( Y, X, n, x):

if (x<X[0]) : return None

if (x>=X[-2]-0.001) : return None

i = 0

while (x > X[i]+0.0001) : i+=1

i-=1

print("segment", [X[i], X[i+1] ], ", x =", x )

if (n==1):

return (Y[i+1]-Y[i])/(X[i+1]-X[i]) + ( ((Y[i+2]-Y[i+1])/(X[i+2]-X[i+1]) - (Y[i+1]-Y[i])/(X[i+1]-X[i]))/(X[i+2]-X[i]) ) \* (2\*x-X[i]-X[i+1])

elif (n == 2):

return 2\*(((Y[i+2] - Y[i+1]) / (X[i+2] - X[i+1]) - (Y[i+1] - Y[i]) / (X[i+1] - X[i])) / (X[i+2] - X[i]));

else:

return None

def diff1(Y, X, n):

return lambda x: Differetiation( Y, X, n, x)

def main():

print("table function: ")

print(Xi)

print(Yi)

d1 = diff1(Yi, Xi, 1)

d2 = diff1(Yi, Xi, 2)

print("diff1", d1(X0) )

print("diff1", d2(X0) )

return 0

main()

## 5 Численное интегрирование

#! /usr/bin/python3.1

# -\*- coding: utf-8 -\*-

from math import sqrt,sin,cos,atan,pi,log

from copy import copy, deepcopy

segment = [-1.0, 1.0]

def func(x):

return x/(3\*x + 4)\*\*2

def func2(x):

return x/(2\*x + 5)

def roundx(v):

return [round(xx, 4) for xx in v]

def P\_n(A):

return lambda x: sum([ A[i]\*x\*\*i for i in range(0,len(A)) ] )

def MethodOfRectangles(f, seg, h):

n = round((seg[1] - seg[0])/h )

return sum( [ h \* f( seg[0] + h\*i + h/2) for i in range(n)] )

def MethodOfTrapezoids(f, seg, h):

n = round((seg[1] - seg[0])/h )

y = [ f(seg[0] + h\*i) for i in range(n+1)]

return sum( [ (y[i] + y[i+1])\*h/2 for i in range(n)] )

def SimpsonMethod(f, seg, h):

n = round((seg[1] - seg[0])/h )

n2 = round((seg[1] - seg[0])/h)//2

y = [ f(seg[0] + h\*i) for i in range(n+1)]

return sum( [( (y[2\*i] + 4\*y[2\*i+1]+y[2\*(i+1)] )\*h/3 )

for i in range(n2)] )

def RRR(f1, f2, k, p):

""" Runge-Romberga-Richardsona """

# k = h1/h2, p - interpolation level

return f1 + (f1 - f2) / (k\*\*p - 1)

def main():

r1 = MethodOfRectangles(func, segment, 0.5)

print("Rect 0.5:\t", r1)

t1 = MethodOfTrapezoids(func, segment, 0.5)

print("Trap 0.5:\t", t1)

s1 = SimpsonMethod (func, segment, 0.5)

print("Simspson 0.5:\t", s1)

r2 = MethodOfRectangles(func, segment, 0.25)

print("Rect 0.25:\t", r2)

t2 = MethodOfTrapezoids(func, segment, 0.25)

print("Trap 0.25:\t", t2)

s2 = SimpsonMethod (func, segment, 0.25)

print("Simspson 0.25:\t", s2)

print("RRR of Rect:\t", RRR(r2, r1, 2, 2) )

print("RRR of Trap:\t", RRR(t2, t1, 2, 2) )

print("RRR of Simpson:\t", RRR(s2, s1, 2, 2) )

print("answer\t", SimpsonMethod(func, segment, 0.0001) )

return 0

main()

## 6 Построение графика табличной функции

#! /usr/bin/python2.6

# -\*- coding: utf-8 -\*-

import numpy as np

import matplotlib.pyplot as plt

from math import sqrt,sin,cos,atan,pi,log

func = lambda x: cos(x)

Xi = [0.1\*pi, 0.2\*pi, 0.3\*pi, 0.4\*pi]

Yi = [sin(xx) for xx in Xi]

X = np.arange(Xi[0], Xi[-1], 0.01)

Y = [ func(xx) for xx in X]

plt.plot(X,Y,color="red")

plt.plot(Xi,Yi,color="grey")

plt.show()

# Протокол тестирования

## 1 Интерполяция многочленом

oleg@debian:~/lab3\_release$ ./lab3\_1.py

table function:

[0.3141592653589793,0.6283185307179586,0.9424777960769379,1.2566370614359172]

[0.3090169943749474,0.5877852522924731,0.8090169943749475,0.9510565162951535]

polynom koef:

0.30902

0.88735

-0.29148

-0.1164

Test lagranje 0.809016994375 = 0.809016994375

Test newton 0.809016994375 = 0.809016994375

|sin(x) - P\_n(x)| = 0.000160111853005

## 2 Интерполяция сплайнами

oleg@debian:~/lab3\_release$ ./lab3\_2.py

table function:

[0.0, 1.0, 2.0, 3.0, 4.0]

[0.0, 1.8415, 2.9093, 3.1411, 3.2432]

Tridiagonal\_Matrix:

[0, 1.0, 1.0]

[4.0, 4.0, 4.0]

[1.0, 1.0, 0]

[-2.3211, -2.508, -0.3891]

x = [0.0, -0.44953, -0.52299, 0.03347]

proverka:

A = [0.0, 1.8415, 2.9093, 3.1411]

B = [1.99134, 1.54181, 0.5693, 0.07979]

C = [0.0, -0.44953, -0.52299, 0.03347]

D = [-0.14984, -0.02449, 0.18549, -0.01116]

test1 2.4969 = 2.49696428571

## 3 Метод наименьших квадратов

oleg@debian:~/lab3\_release$ ./lab3\_3.py

table function:

[0.0, 1.7, 3.4, 5.1, 6.8, 8.5]

[0.0, 1.3038, 1.8439, 2.2583, 2.6077, 2.9155]

Matrix 3x3 :

6.0 25.5 158.9

25.5 158.9 1105.4

158.9 1105.4 8176.7

-----

b = [10.9292, 62.5172, 415.0468]

p = [2, 1, 0]

Ai = [0.1294, 0.6193, -0.0355]

test1 1.3038 = 0.129442857143

F\_bolshoe = 0.0945576948571

## 4 Численное дифференцирование

oleg@debian:~/lab3\_release$ ./lab3\_4.py

table function:

[0.0, 0.1, 0.2, 0.3, 0.4]

[1.0, 1.1052, 1.2214, 1.3499, 1.4918]

segment [0.1, 0.2] , x = 0.2

diff1 1.2235

segment [0.1, 0.2] , x = 0.2

diff1 1.23

## 5 Численное интегрирование

oleg@debian:~/lab3\_release$ ./lab3\_5.py

Rect 0.5: -0.119143132913

Trap 0.5: -0.276633496374

Simspson 0.5: -0.205579355709

Rect 0.25: -0.149311959381

Trap 0.25: -0.197888314644

Simspson 0.25: -0.171639920734

RRR of Rect: -0.15936823487

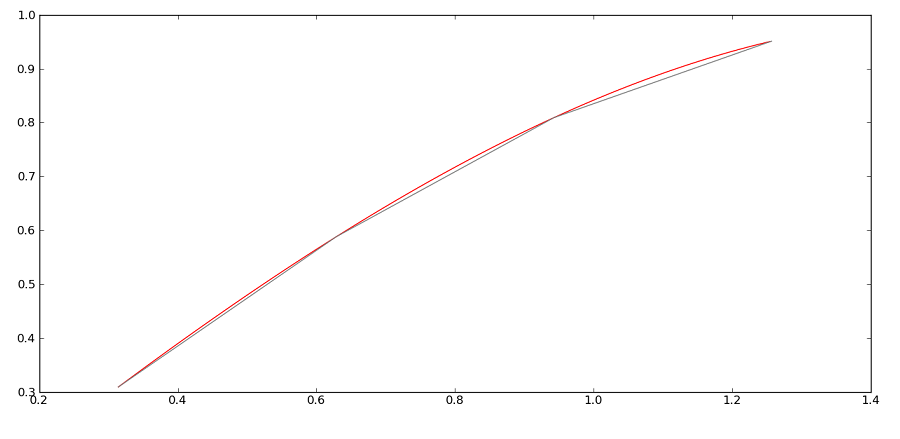
RRR of Trap: -0.171639920734

RRR of Simpson: -0.160326775742

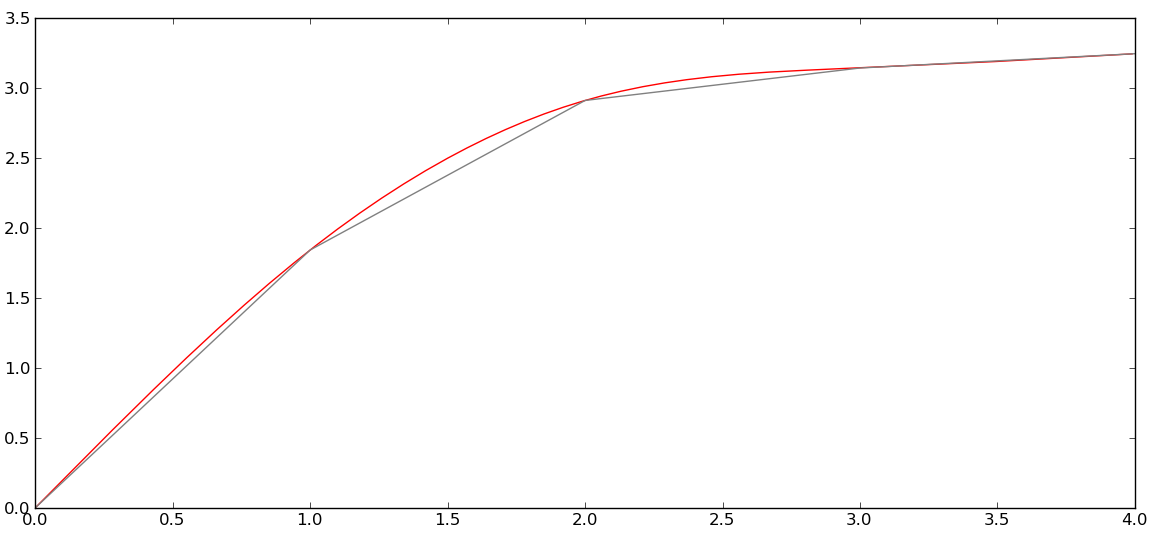
answer -0.164740142168

# Графики

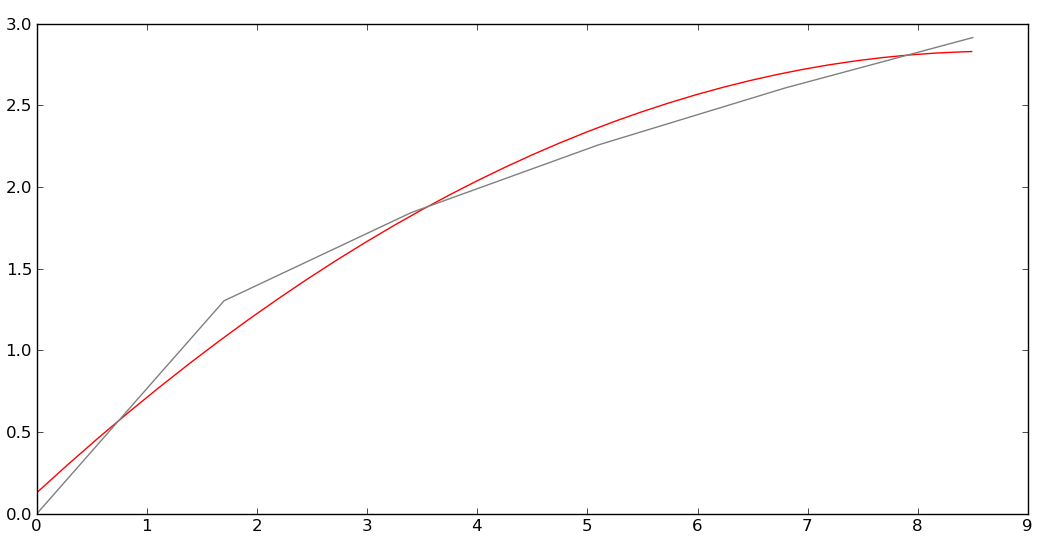
## 1 Интерполяция многочленом



## 2 Интерполяция сплайнами



## 3 Метод наименьших квадратов



# Выводы

## 1 Интерполяция многочленом

Используя таблицу значений функции , вычисленных в точках , построить интерполяционные многочлены Лагранжа и Ньютона, проходящие через точки }. Вычислить значение погрешности интерполяции в точке .

## 2 Интерполяция сплайнами

Построить кубический сплайн для функции, заданной в узлах интерполяции, предполагая, что сплайн имеет нулевую кривизну при и . Вычислить значение функции в точке .

## 3 Метод наименьших квадратов

Для таблично заданной функции путем решения нормальной системы МНК найти приближающие многочлены a) 1-ой и б) 2-ой степени. Для каждого из приближающих многочленов вычислить сумму квадратов ошибок. Построить графики приближаемой функции и приближающих многочленов.

## 4 Численное дифференцирование

Вычислить первую и вторую производную от таблично заданной функции в точке .

## 5 Численное интегрирование

Вычислить определенный интеграл методами прямоугольников, трапеций, Симпсона с шагами . Оценить погрешность вычислений, используя Метод Рунге-Ромберга.