МИНИСТЕРСТВО НАУКИ И ОБРАЗОВАНИЯ РФ

ПЕРМСКИЙ ГОСУДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

Отчет

По лабораторной работе №5 «Интерполирование. Среднеквадратичное приближение. Равномерное приближение»



Пермь 2023

**Задание**

*Задача* - приблизить заданную функцию на отрезке .



1) Построить таблицу , .



По полученной таблице произвести интерполяцию с помощью

* + формулы Ньютона;
  + кубических сплайнов дефекта 1.

Провести оценку погрешности в узлах .



Сравнить оценку погрешности с реальной погрешностью.

2) Выполнить среднеквадратичное приближение заданной функции на заданном отрезке c помощью полинома второго порядка

* + дискретный вариант (по таблице из п.1);
  + непрерывный (интегральный) вариант.

Провести оценку погрешности по норме. Вычислить погрешность в узлах интерполяции.

3) Выполнить равномерное приближение заданной функции на заданном отрезке с помощью полинома первого порядка. Провести оценку погрешности по норме. Вычислить погрешность в узлах интерполяции.

4) Методом обратного интерполирования, используя интерполяционную формулу Ньютона, найти корень уравнения *f(x)=c,* где .



*Указание.*

1. При построении параметров кубического сплайна воспользоваться алгоритмом прогонки. Выполнить оценку погрешности аппроксимации значений первой производной в узлах интерполяции.
2. При решение СЛАУ для задачи среднеквадратичного приближения воспользоваться методом квадратного коря. В непрерывном (интегральном) варианте среднеквадратичного приближения все интегралы считать с максимально возможной точностью.

**Исходные данные**

Вариант 1.

image001

Вариант 23.

image021

**Теоретическая справка**

**Интерполяционная формула Ньютона**

Ряд Тейлора

f(x) = f(x0) + (x – x0) f'(x0) + (x – x0)2 +….



Система базисных функций

φ0(x) = ω0(x) = 1, φ1(x) = ω1(x) = (x –x0),

φ2(x) = ω2(x) = (x –x0)(x –x1), …, φn(x) = ωn(x).

Очевидное свойство

ωk(xi) = 0 для k =, i=.



Интерполяционный многочлен

Pn(x) = .

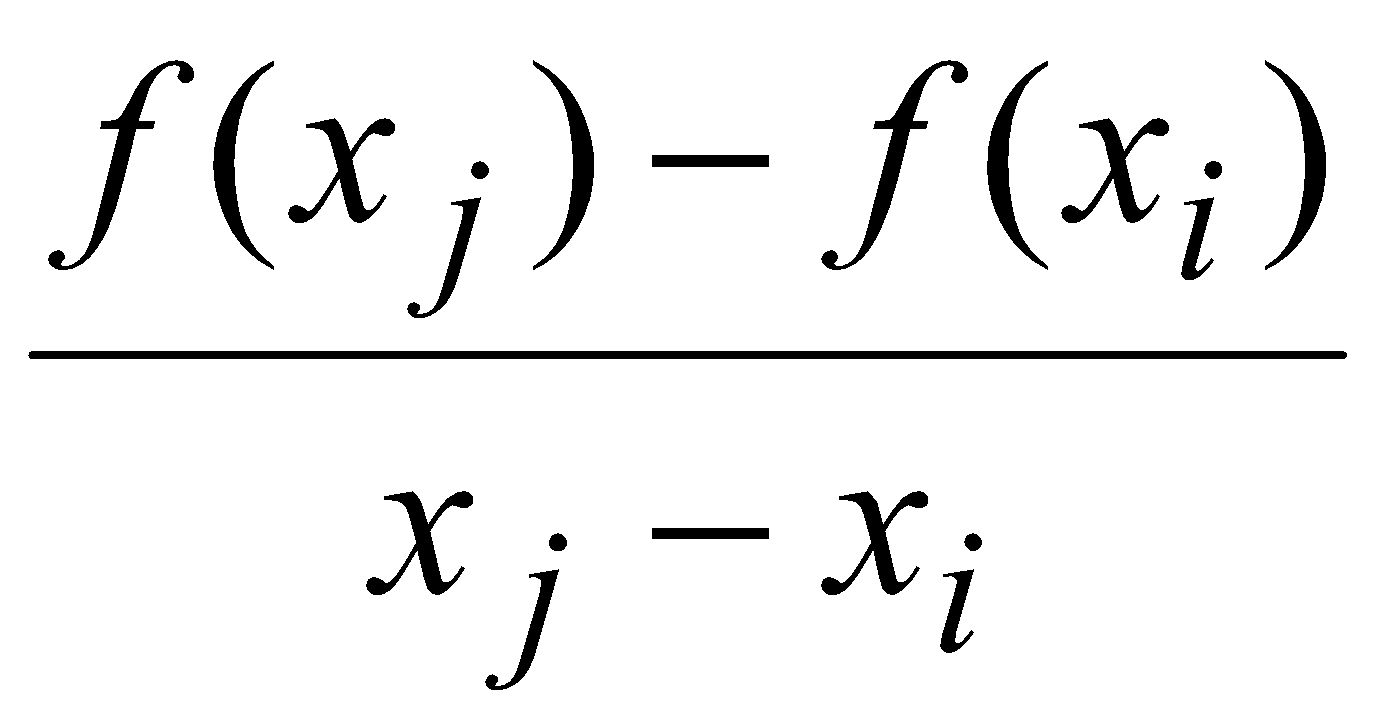
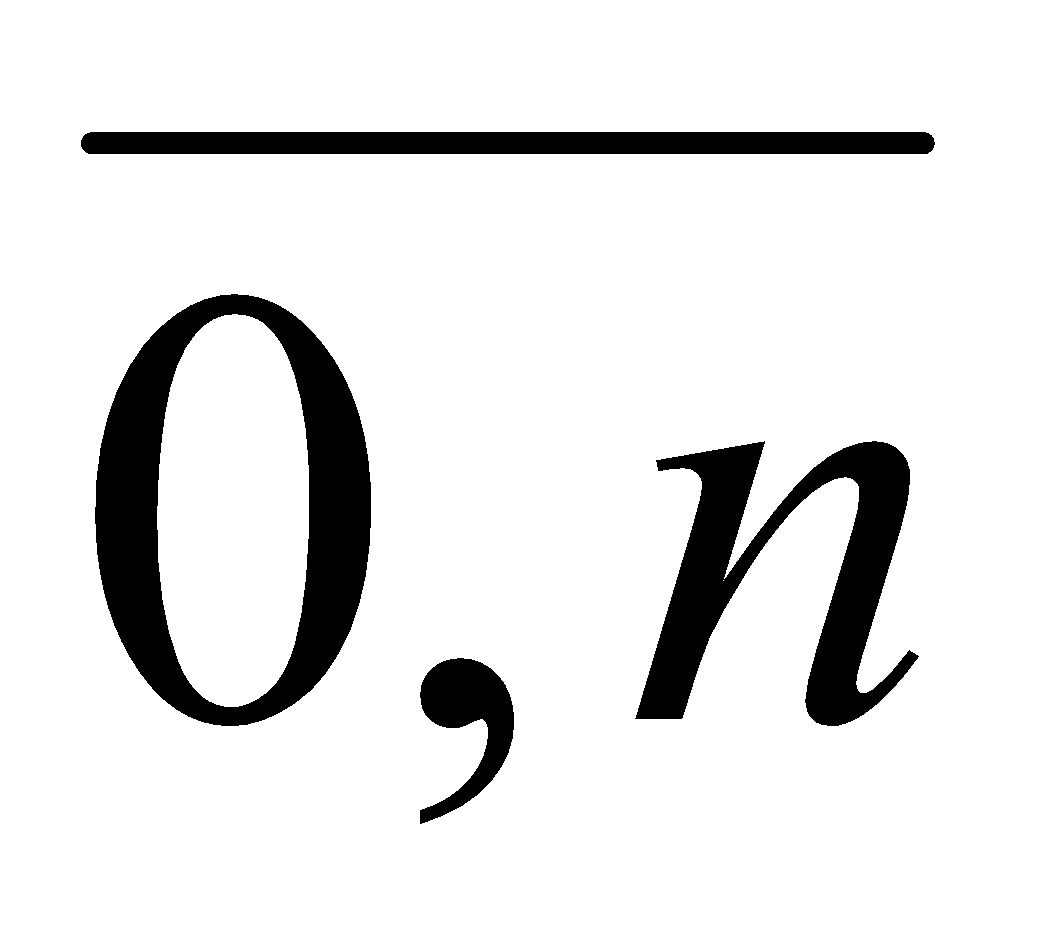


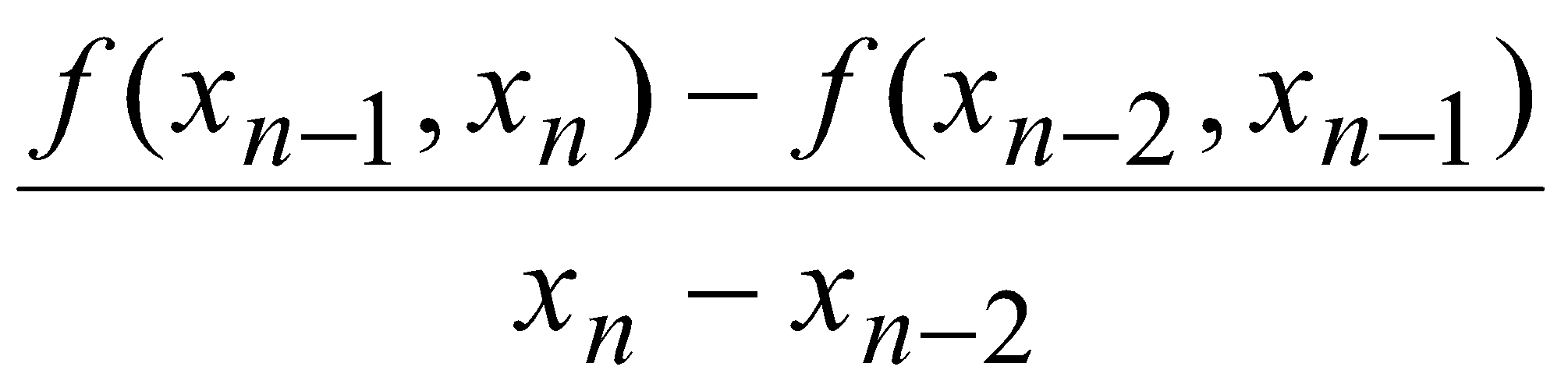
Условия интерполяции



Имеем относительно коэффициентов ck СЛАУ с нижней треугольной матрицей

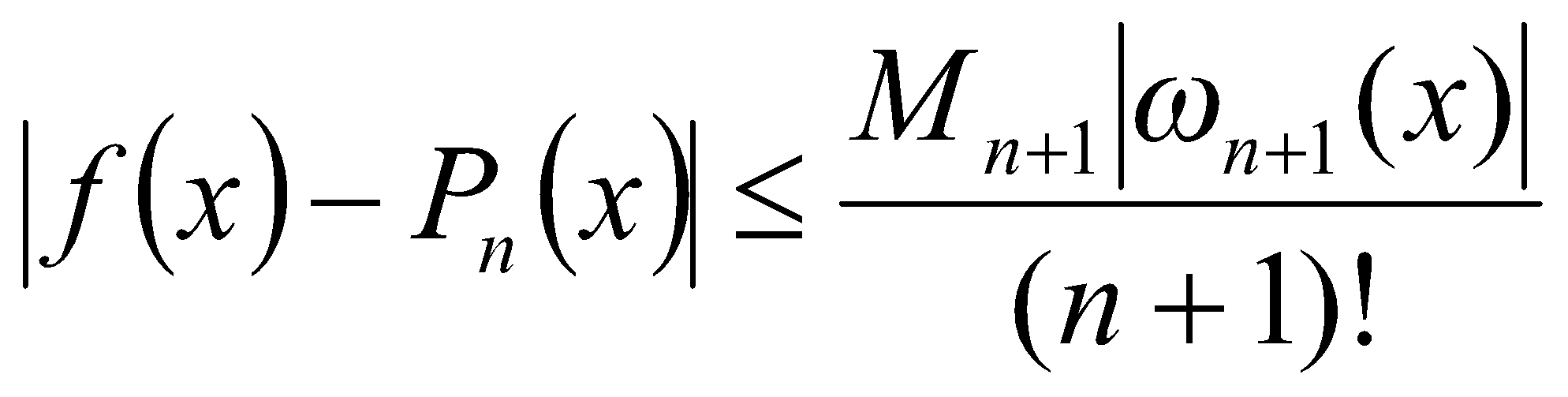
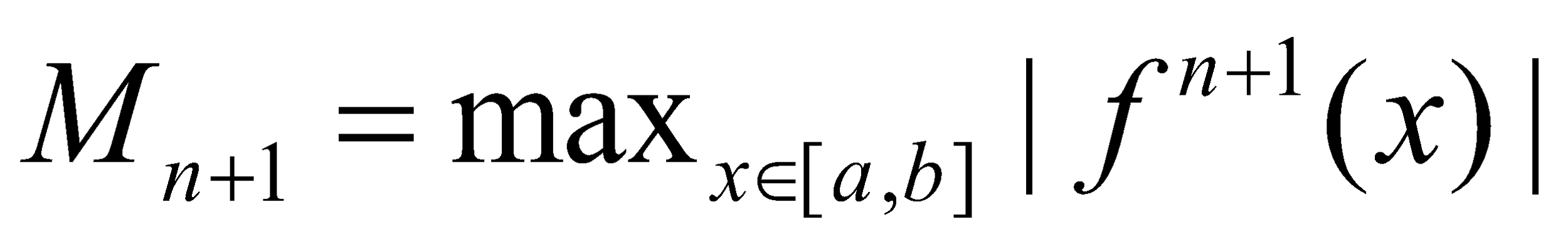
Для таблицы разделенных разностей используем формулу:

*f*(*xi*, *xj*) = , *i*, *j* = , *i* ≠ *j*.

*f*(*xn*-2, *xn*-1, *xn*) = 

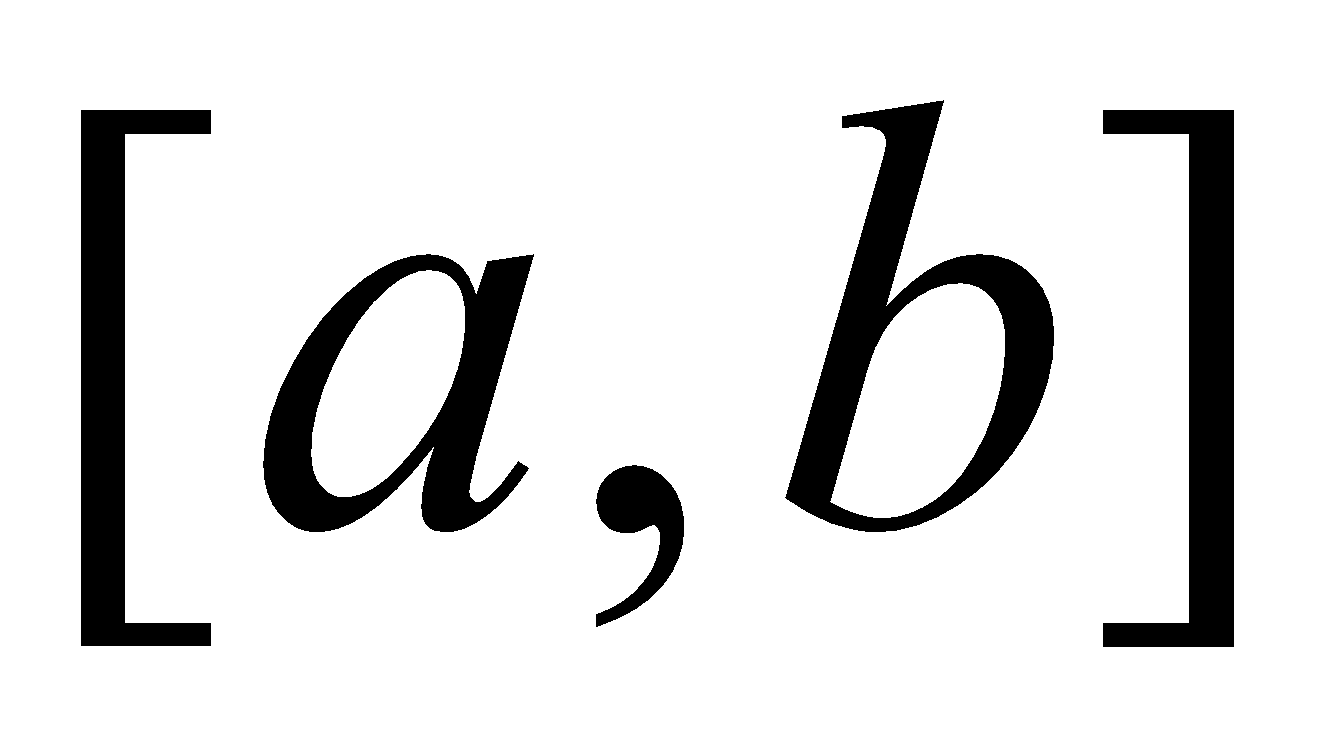
Поскольку интерполяционные формулы Лагранжа и Ньютона отличаются только формой записи, представление погрешности справедливо и для формулы Ньютона.

Для оценки погрешности используем формулу:

, где 

В нашем случае n=5, значит рассчитываем M6

**Интерполяционный кубический сплайн дефекта 1**

*Сплайн-функцией* или *сплайном* называют кусочно-полиномиальную функцию, определенную на отрезке  и имеющую на этом отрезке некоторое число непрерывных производных.

Алгоритм:

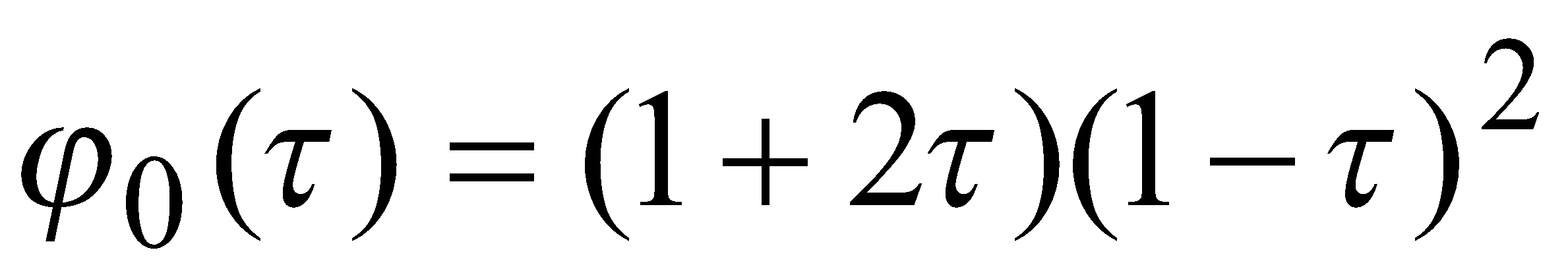
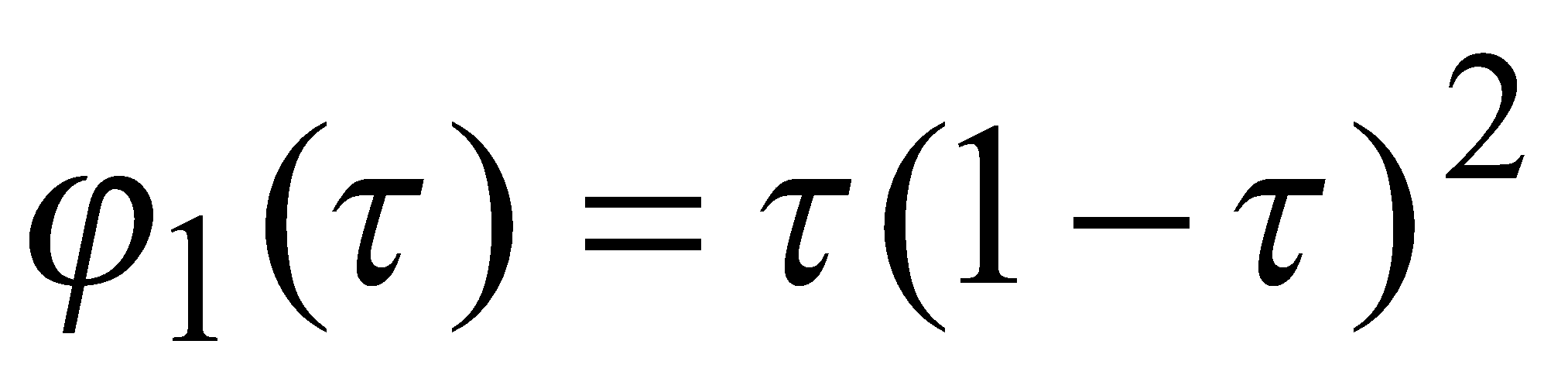
Решая СЛАУ методом прогонки находим величины mi , . Затраты .

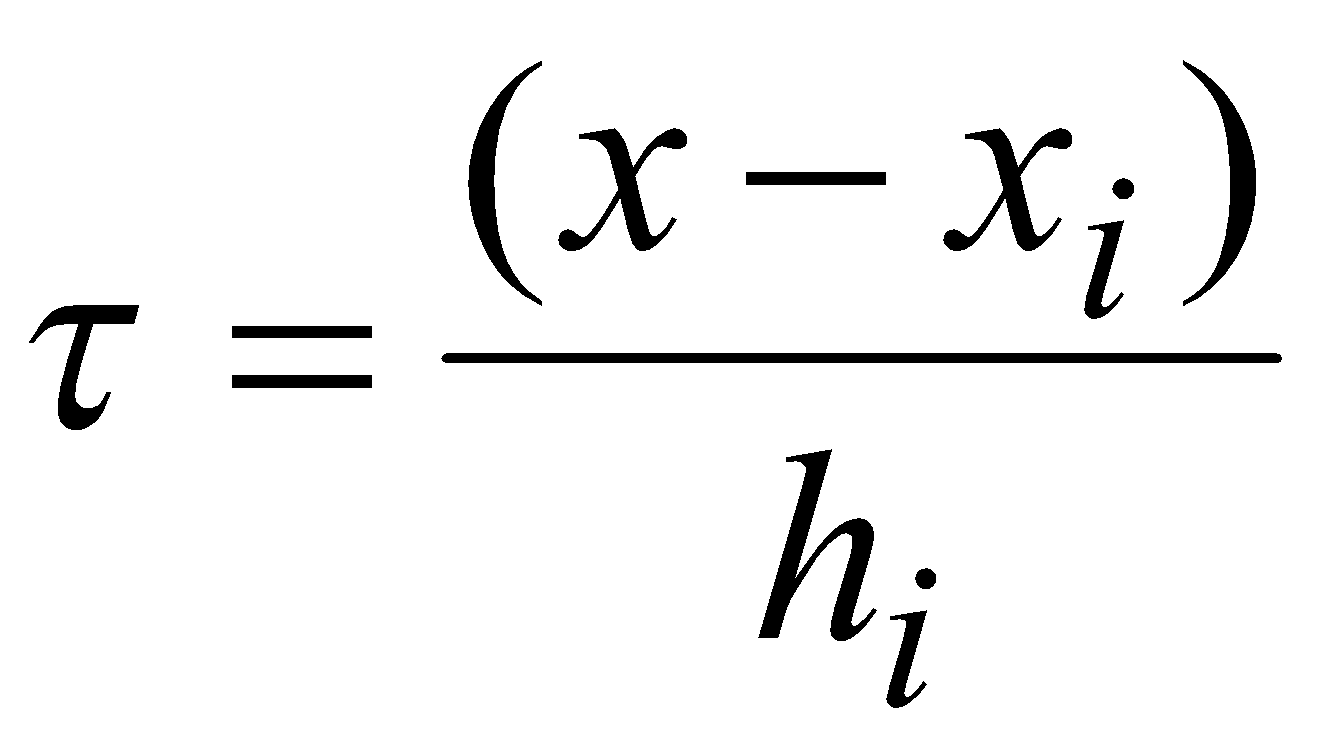
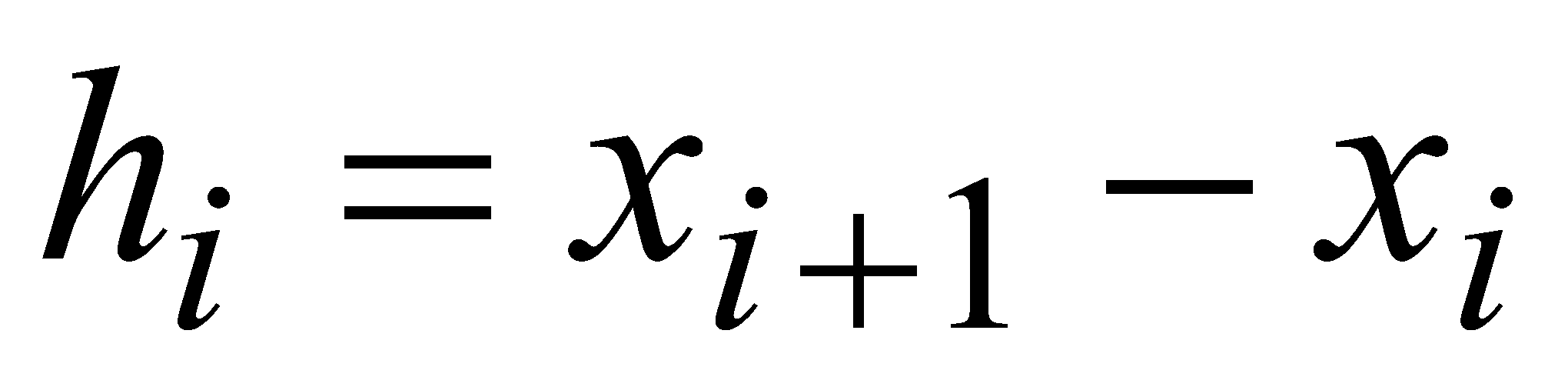


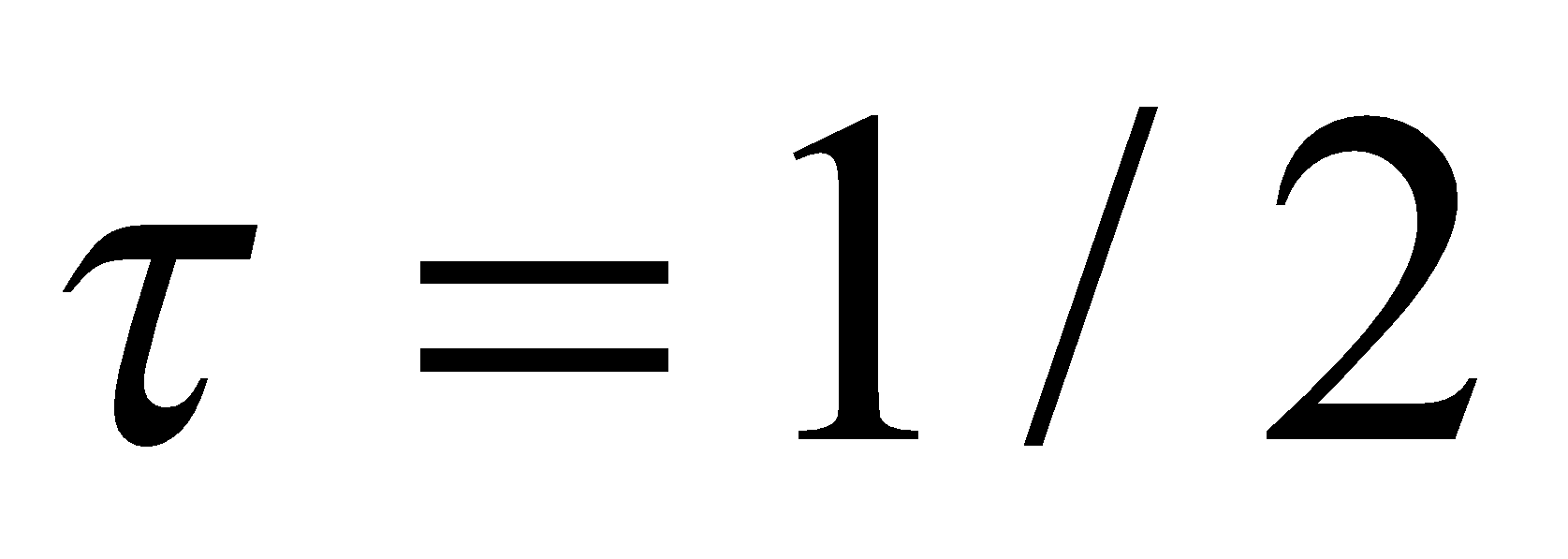
Определив отрезок , которому принадлежит точка х, вычисляем



*S3,1(f; x) = ϕ0(τ)fi + ϕ0(1-τ)fi+1 + hi(ϕ1(τ)mi - ϕ1(1-τ)mi + 1)*

где , ,

, .

В нашем случае h=1/5, 

Погрешность интерполяции:

В случае равномерной сетки оценки могут быть улучшены:

|– mi| ≤,



**Среднеквадратичное приближение.**

Для этого используем матрицу скалярных произведений 3\*3:

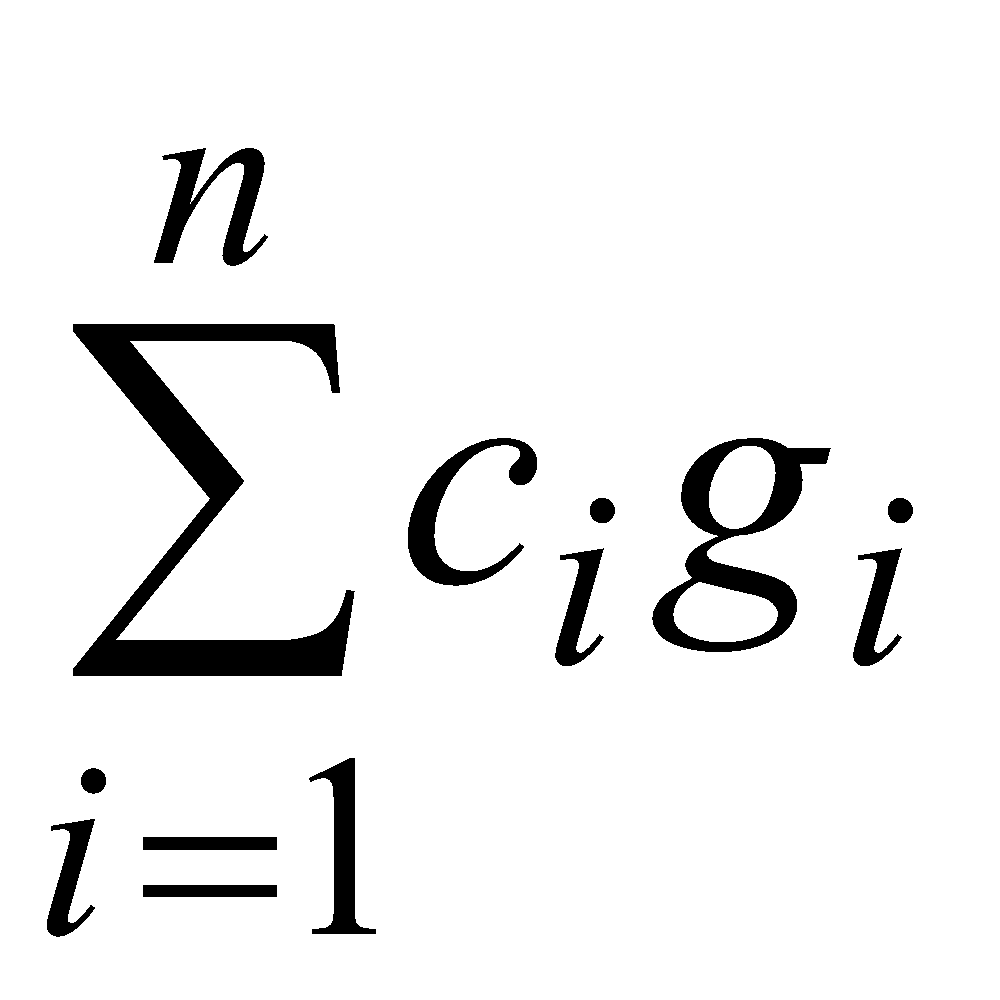
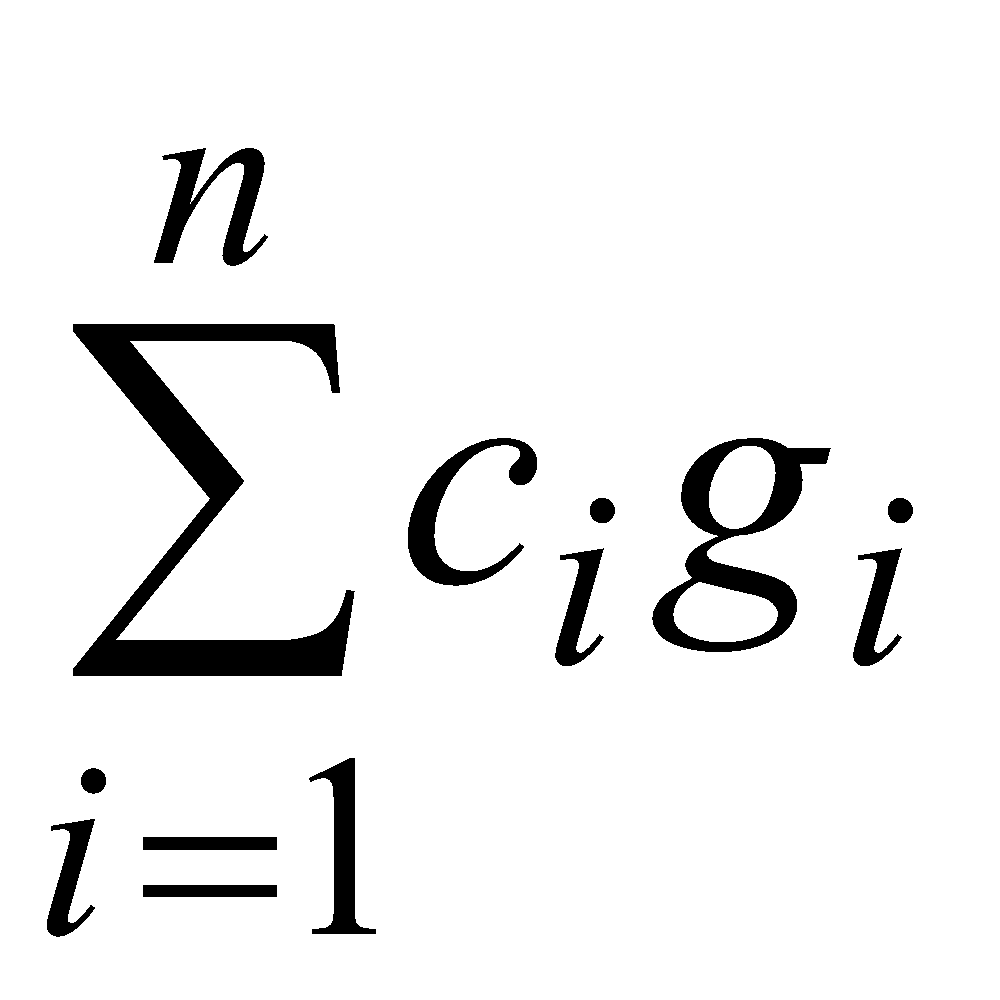
1. Дискретный вариант:

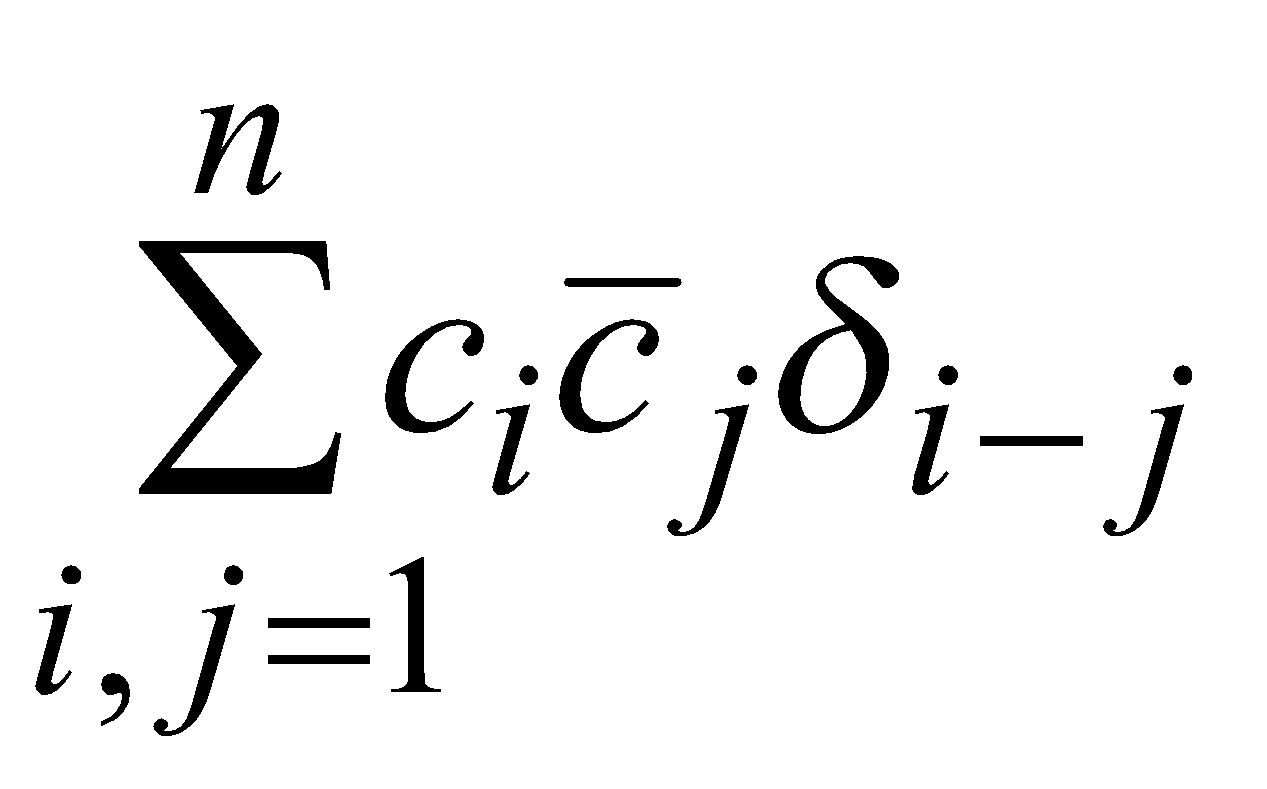
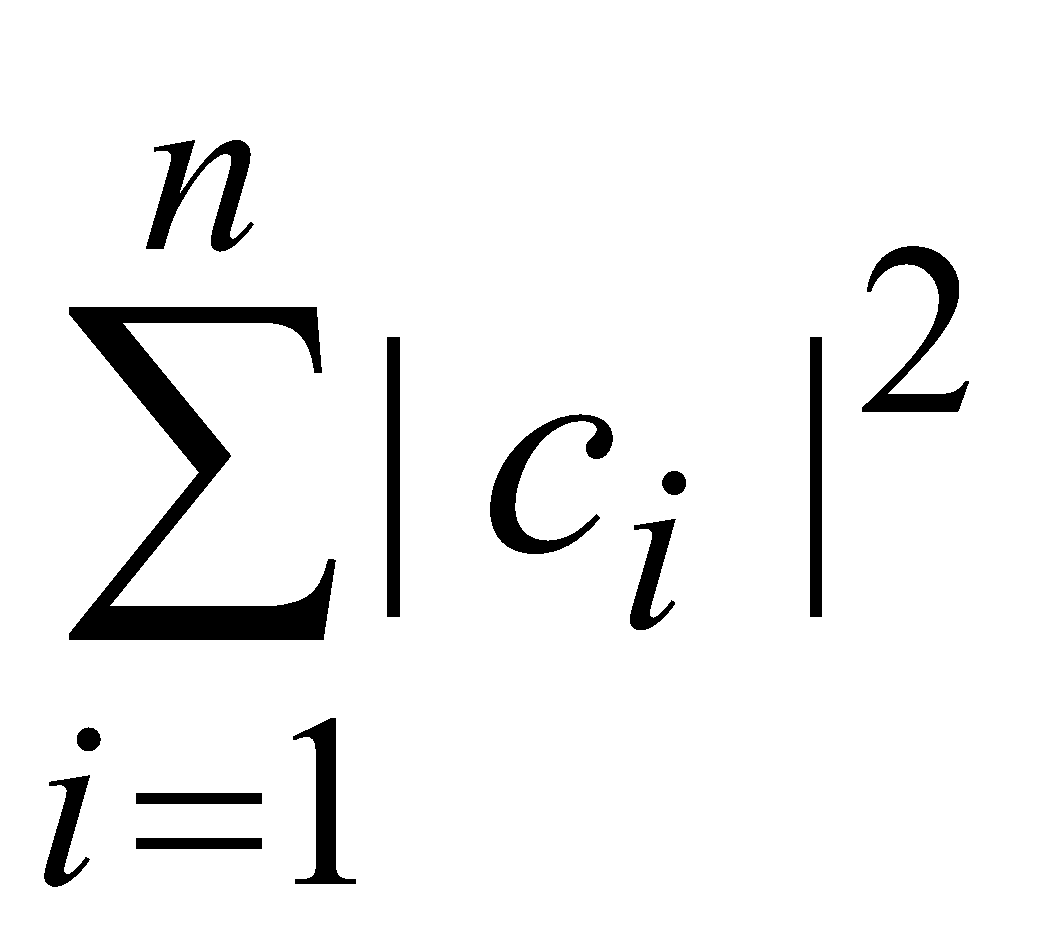
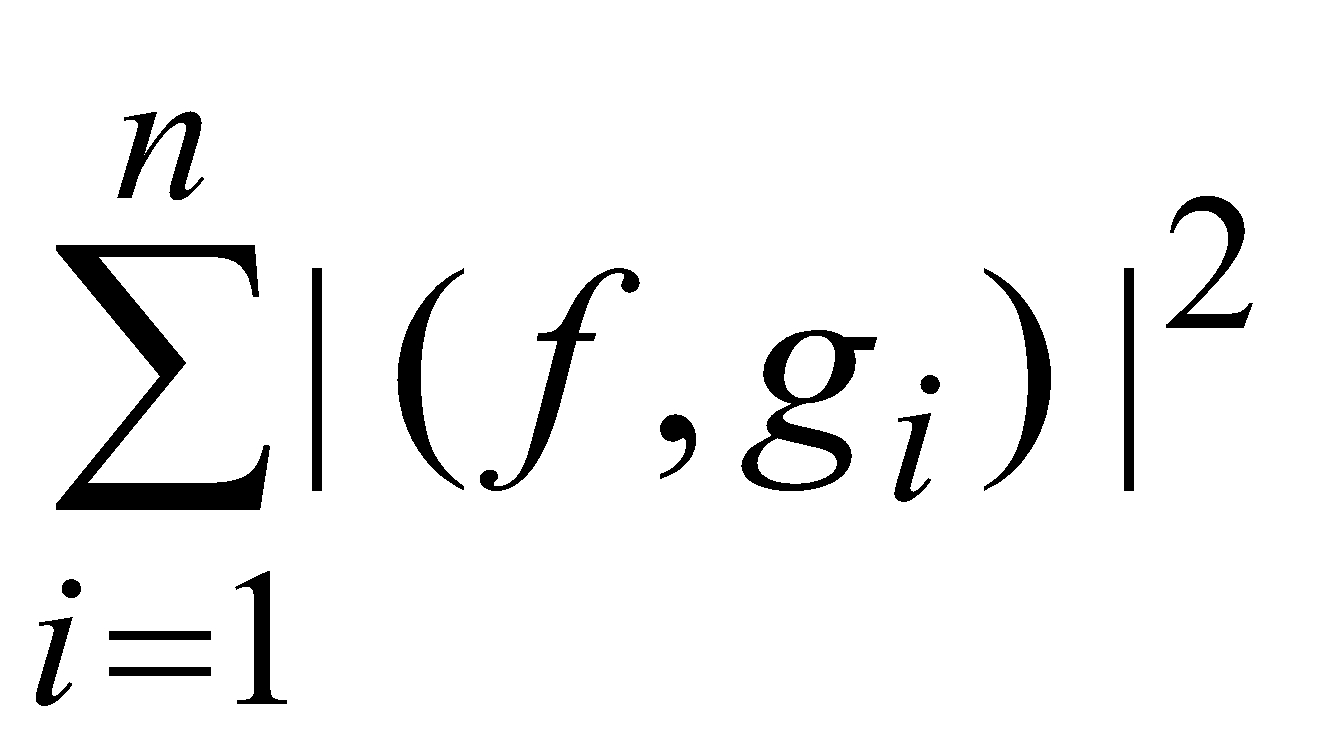
Матрица считается с узлами, возведенными в степень.

Находим вектор правых частей.

Далее решаем СЛАУ и находим вектор коэффициентов полинома Pn(x).

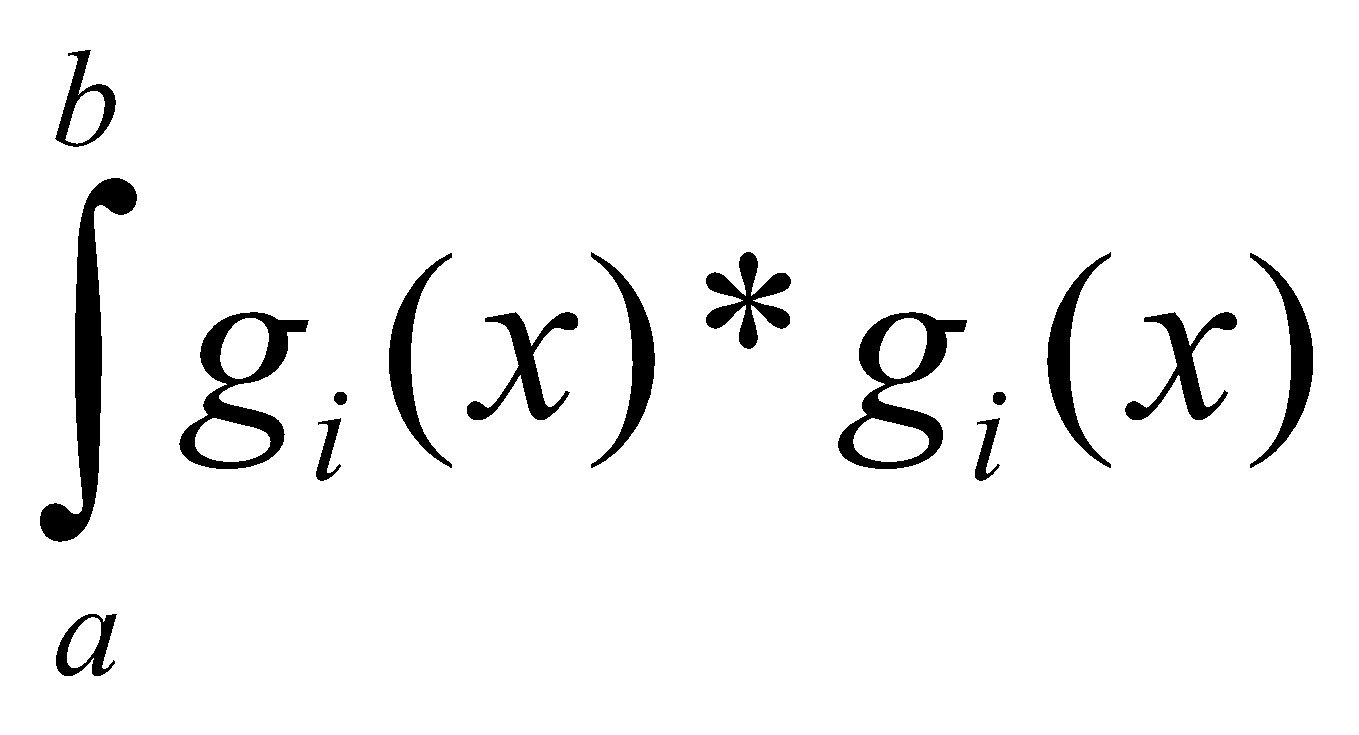
Для оценки используем формулу:

||*f* – *g*||2 = ||*f*||2 - ||*g*||2 = (*f*, *f*) – (, ) =

= (*f*, *f*) –= (*f*, *f*) – = (*f*, *f*) – 

1. Непрерывный вариант:

Находим матрицу скалярных произведений, где каждый элемент матрицы:



Находим вектор правых частей с помощью интегралов произведения функции и базиса.

Решаем СЛАУ и находим вектор коэффициентов для полинома Pn(x).

**Равномерное приближение.**

Функция f(x) – (а0 + а1х) имеет только одну внутреннюю точку экстремума. (вследствие выпуклости f(x)), обозначим ее d.

Тогда a, b, d - точки Чебышевского альтернанса.

По теореме имеем ():



f(a) – (а0 + а1 ⋅ a) = αL,

f(d) – (а0 + а1 ⋅ d) = – αL,

f(b) – (а0 + а1 ⋅ b) = αL.

и

f '(d) – a1 = 0.

Учитывая, что

f(b) – f(a) = a1(b – a)

f(а) + f(d) – 2a0 = a1(a + d)

Решение этой системы

a1 = , f '(d) =,



a0 = .



Метод обратной интерполяции.

.



**Нахождение корней трансцендентных уравнений методом обратного интерполирования**

Пусть требуется определить корень уравнения



и удалось выделить (например, графически) отрезок , на котором функция монотонно возрастает (или убывает) и содержит корень.



Выберем на этом отрезке точки и вычислим значения . В силу монотонности функции на отрезке имеем (или ). Тогда мы можем построить интерполяционный многочлен , для которого .



Очевидно , где - корень уравнения. При этом погрешность решения можно определить с помощью формулы



Mn+1 =|f (n+1)(y)|.



**Решение**

**Вариант № 1**

Интерполяционная формула Ньютона

Таблица разделенных разностей

1.00 4.000000 6.486984 0.552780 0.136996 0.025464 0.003786

1.20 5.297397 6.708096 0.634977 0.157367 0.029250

1.40 6.639016 6.962087 0.729398 0.180767

1.60 8.031433 7.253846 0.837858

1.80 9.482202 7.588989

2.00 11.000000

M6 = 0.443622

x f(x) Pn(x) Delta Оценка

1.10 4.643547 4.643547 3.971572e-07 5.822534e-07

1.30 5.962289 5.962289 1.349734e-07 1.940845e-07

1.50 7.328427 7.328427 9.832226e-08 1.386318e-07

1.70 8.749010 8.749009 1.404233e-07 1.940845e-07

1.90 10.232132 10.232132 4.298832e-07 5.822534e-07

Интерполяция кубическим сплайном

M5 = 0.640011

x[i] df/dx(x[i]) m[i] Delta Оценка

1.00 6.386294 6.386294 0.000000e+00 1.706695e-05

1.20 6.592434 6.592430 3.993258e-06 1.706695e-05

1.40 6.829226 6.829223 3.649700e-06 1.706695e-05

1.60 7.101229 7.101225 3.948540e-06 1.706695e-05

1.80 7.413679 7.413672 6.448490e-06 1.706695e-05

2.00 7.772589 7.772589 0.000000e+00 1.706695e-05

M4 = 0.92334

x f(x) S31(f;x) Abs(f(x)-S31(f;x)) Оценка

1.10 4.643547 4.643545 1.962520e-06 4.700599e-06

1.30 5.962289 5.962286 2.377609e-06 4.700599e-06

1.50 7.328427 7.328424 2.713818e-06 4.700599e-06

1.70 8.749010 8.749007 3.063442e-06 4.700599e-06

1.90 10.232132 10.232128 3.751975e-06 4.700599e-06

Среднеквадратичное приближение

Дискретный вариант

Матрица

6.0000 9.0000 14.2000

9.0000 14.2000 23.4000

14.2000 23.4000 39.9664

Вектор правых частей

44.4500479177451524 71.5697552707223537 119.9235263934722155

P2(x) = -1.61493 + (4.93177)\*x + (0.68688)\*x^2

Норма погрешности: 0.01020

x Погрешность

1.00 -0.0037225392365041

1.20 0.0050935737610454

1.40 0.0031819019392216

1.60 -0.0028817563363379

1.80 -0.0055437923002266

2.00 0.0038726120106674

Непрерывный вариант

Матрица

1.0000 1.5000 2.3333

1.5000 2.3333 3.7500

2.3333 3.7500 6.2000

Вектор правых частей

7.3853900817779303 11.6600989499891998 18.9824256290057995

P2(x) = -1.60602 + (4.92823)\*x + (0.68531)\*x^2

Норма погрешности: 0.00299

x Погрешность

1.00 -0.0075258044712099

1.20 0.0026876591781075

1.40 0.0022985547626284

1.60 -0.0021173193520116

1.80 -0.0030063544004015

2.00 0.0083082675805652

Равномерное приближение

P1(x)= -3.08607 + 7.00000\*x, d = 1.52877

L(a)=0.08607, L(d)=-0.08607, L(b)=0.08607

x Погрешность

1.00 0.0860713320559343

1.20 -0.0165319579499954

1.40 -0.0749128463982753

1.60 -0.0824955349232699

1.80 -0.0317264147595697

2.00 0.0860713320559334

Решение уравнения методом обратной интерполяции

Таблица разделенных разностей

-3.500000 1.000000 0.154155 -0.001925 -0.000016 0.000001 0.000000

-2.202603 1.200000 0.149074 -0.001989 -0.000010 0.000001

-0.860984 1.400000 0.143635 -0.002032 -0.000004

0.531433 1.600000 0.137858 -0.002051

1.982202 1.800000 0.131770

3.500000 2.000000

c = 7.500000

Корень 1.524590

Невязка = Abs(f(x) - c) = 0.000000174097

**Вариант № 23**

Интерполяционная формула Ньютона

Таблица разделенных разностей

1.00 2.000000 5.685964 2.264389 0.927384 0.284859 0.069999

1.20 3.137193 6.591720 2.820819 1.155271 0.354857

1.40 4.455537 7.720047 3.513981 1.439157

1.60 5.999546 9.125640 4.377475

1.80 7.824674 10.876630

2.00 10.000000

M6 = 15.823743

x f(x) Pn(x) Delta Оценка

1.10 2.548370 2.548381 1.136400e-05 2.076866e-05

1.30 3.771168 3.771164 3.904375e-06 6.922888e-06

1.50 5.196152 5.196155 2.876567e-06 4.944920e-06

1.70 6.873008 6.873004 4.156939e-06 6.922888e-06

1.90 8.863626 8.863639 1.288243e-05 2.076866e-05

Интерполяция кубическим сплайном

M5 = 14.4034

x[i] df/dx(x[i]) m[i] Delta Оценка

1.00 5.295837 5.295837 0.000000e+00 3.840904e-04

1.20 6.105726 6.105662 6.361932e-05 3.840904e-04

1.40 7.114630 7.114565 6.523904e-05 3.840904e-04

1.60 8.371453 8.371379 7.370502e-05 3.840904e-04

1.80 9.937116 9.936980 1.360912e-04 3.840904e-04

2.00 11.887511 11.887511 0.000000e+00 3.840904e-04

M4 = 13.1105

x f(x) S31(f;x) Abs(f(x)-S31(f;x)) Оценка

1.10 2.548370 2.548351 1.874944e-05 7.383174e-05

1.30 3.771168 3.771142 2.529758e-05 7.383174e-05

1.50 5.196152 5.196121 3.135277e-05 7.383174e-05

1.70 6.873008 6.872970 3.776113e-05 7.383174e-05

1.90 8.863626 8.863574 5.238539e-05 7.383174e-05

Среднеквадратичное приближение

Дискретный вариант

Матрица

6.0000 9.0000 14.2000

9.0000 14.2000 23.4000

14.2000 23.4000 39.9664

Вектор правых частей

33.4169497312299910 55.6860699092485660 95.9611916797656193

P2(x) = 0.52792 + (-1.72292)\*x + (3.22223)\*x^2

Норма погрешности: 0.07554

x Погрешность

1.00 -0.0272331665282550

1.20 0.0367611130033421

1.40 0.0241278444761726

1.60 -0.0206185460130595

1.80 -0.0420250606164512

2.00 0.0289878157795513

Непрерывный вариант

Матрица

1.0000 1.5000 2.3333

1.5000 2.3333 3.7500

2.3333 3.7500 6.2000

Вектор правых частей

5.4614353597610297 8.8490423679278898 14.7318165914863002

P2(x) = 0.57921 + (-1.72902)\*x + (3.20390)\*x^2

Норма погрешности: 0.02212

x Погрешность

1.00 -0.0540824056187827

1.20 0.0191997202566352

1.40 0.0173210827185235

1.60 -0.0152038921364115

1.80 -0.0229222064602697

2.00 0.0632456548605038

Равномерное приближение

P1(x)= -6.40525 + 8.00000\*x, d = 1.54532

L(a)=0.40525, L(d)=-0.40525, L(b)=0.40525

x Погрешность

1.00 0.4052535152065850

1.20 -0.0575536659468625

1.40 -0.3392097630473359

1.60 -0.3952003499981274

1.80 -0.1700724289513396

2.00 0.4052535152065850

Решение уравнения методом обратной интерполяции

Таблица разделенных разностей

-4.000000 1.000000 0.175872 -0.009842 0.000524 -0.000023 0.000001

-2.862807 1.200000 0.151705 -0.007746 0.000389 -0.000017

-1.544463 1.400000 0.129533 -0.005922 0.000273

-0.000454 1.600000 0.109581 -0.004410

1.824674 1.800000 0.091940

4.000000 2.000000

c = 6.000000

Корень 1.600054

Невязка = Abs(f(x) - c) = 0.000000003058

**Вывод**

Были программно реализованы методы интерполяции функции с помощью формулы Ньютона и кубических сплайнов дефекта 1, также выполнено среднеквадратичное приближения заданной функции на заданном отрезке c помощью полинома второго порядка (дискретный и непрерывные варианты), равномерное приближение с помощью полинома 1 порядка и рассмотрена задача нахождения корней трансцендентных, нелинейных уравнений методом обратного интерполирования.

В интерполяционной формуле Ньютона берется полином 5-го порядка, поэтому погрешность 10^(-6).

В кубическом сплайне дефекта один берутся полиномы 3-го порядка, но их достаточно много, поэтому погрешность 10^(-5), что не сильно отличается от погрешности в формуле Ньютона.

Интерполяционная формула Ньютона и кубический сплайн дефекта 1 дают довольно точное решение.

В среднеквадратичном приближении берутся полиномы 2-го порядка, поэтому погрешность хуже, чем в перечисленных выше методах. Однако, если сравнивать дискретных вариант и непрерывный вариант, то можно увидеть, что непрерывный вариант точнее.

В равномерном приближении берутся полиномы 1-го порядка, поэтому данный метод самый неточный из всех.

| Метод | Погрешность |
| --- | --- |
| Формула Ньютона | 10^(-6) |
| Кубический сплайн дефекта 1 | 10^(-5) |
| Среднеквадратичное приближение (непрерывный вариант) | 10^(-2) или 10^(-1) |
| Среднеквадратичное приближение (дискретный вариант) | 10^(-2) или 10^(-1) |
| Равномерное приближение | 10^(-1) или 10^(0) |

**Код программы**

#include <windows.h>

#include "iostream"

#include <iomanip>

#include <vector>

#include <fstream>

using namespace std;

ofstream out("output.txt");

const int n = 6;

double b = 2.0;

double a = 1.0;

double h = (b - a) / (n - 1);

double a1, d, a0, aL, bL, dL;

double eps = pow(10, -8);

typedef vector<double> myVector;

double\* yArr = new double[n];

// Исходная функция f(x)

double func(double x, int var)

{

if (var == 1)

return pow(2, x) + 5 \* x - 3;

if (var == 23)

return pow(3, x) + 2 \* x - 3;

return 0;

}

// Производная функции f'(x)

double Derivative(double x, int var)

{

if (var == 1)

return pow(2, x) \* log(2) + 5;

if (var == 23)

return pow(3, x) \* log(3) + 2;

return 0;

}

void get\_Arr(double\* xArr, double\* xArr1, double\* yArr, int var)

{

//Таблица значений xi,i=1,...,n

for (int i = 0; i < n; i++)

{

xArr[i] = 1.0 + i \* h;

}

// Таблица x\*i,i=1,...,n (узлов)

for (int i = 0; i < n - 1; i++)

{

xArr1[i] = 1.0 + (i + 0.5) \* h;

}

//Таблица {xi,f(x)},i=1,...,n

for (int i = 0; i < n; i++)

{

yArr[i] = func(xArr[i], var);

}

}

// Вывод таблицы разделенных разностей для формулы Ньютона

void PrintSplitDifferenceMatrix(double\*\* SplDiff)

{

out << "Таблица разделенных разностей" << endl;

for (int i = 0; i < n; i++)

{

for (int j = 0; j <= n - i; j++)

{

if (j == 0)

{

out << fixed << setprecision(2) << setw(4) << SplDiff[i][j];

}

else

{

out << fixed << setprecision(6) << setw(12) << SplDiff[i][j];

}

}

out << endl;

}

}

// Вывод таблицы разделенных разностей для метода обратной интерполяции

void PrintSplitDifferenceMatrix2(double\*\* SplDiff)

{

out << "Таблица разделенных разностей" << endl;

for (int i = 0; i < n; i++)

{

for (int j = 0; j <= n - i; j++)

{

out << fixed << setprecision(6) << setw(12) << SplDiff[i][j];

}

out << endl;

}

}

// Вычисление таблицы разделённых разностей

void FindDividedDifferenceMatrix(double\*\* SplDiff, double\* xArr, double\* yArr)

{

for (int i = 0; i < n; i++)

{

SplDiff[i][0] = xArr[i];

SplDiff[i][1] = yArr[i];

}

for (int j = 2; j <= n; j++)

{

for (int i = 0; i < n - j + 1; i++)

SplDiff[i][j] = (SplDiff[i + 1][j - 1] - SplDiff[i][j - 1]) / (xArr[i + j - 1] - xArr[i]);

}

}

// Погрешность оценки интерполяции для формулы Ньютона

double FindErrorForNewton(double x, double\* xArr, double M6)

{

double w = 1;

for (int i = 0; i < n; i++)

w \*= x - xArr[i];

int fact = 1;

for (int i = 2; i <= n; i++)

fact \*= i;

return M6 \* abs(w) / fact;

}

// Интерполяционная формула Ньютона

void Newton(double\* xArr, double\* xArr1, double\* yArr, int var)

{

double\*\* SplDiff = new double\* [n];

for (int i = 0; i < n; i++)

{

SplDiff[i] = new double[n + 1];

}

//заполнение таблиц

get\_Arr(xArr, xArr1, yArr, var);

//вычисление таблицы разделенных разностей и ее печать

FindDividedDifferenceMatrix(SplDiff, xArr, yArr);

PrintSplitDifferenceMatrix(SplDiff);

//вычисление максимума 6 производной

double M6 = 0;

if (var == 1)

M6 = pow(2, 2) \* pow(log(2), 6);

else

if (var == 23)

M6 = pow(3, 2) \* pow(log(3), 6);

out << endl << "M6 = " << M6 << endl;

//вывод шапки

out << endl << "x f(x) Pn(x) Delta Оценка" << endl;

//вычисления по формуле Ньютона

for (int k = 0; k < n - 1; k++)

{

double Pn\_x = SplDiff[0][1];//интерполяционный многочлен

double w = 1;

for (int i = 1; i < n; i++)

{

w \*= xArr1[k] - xArr[i - 1];

Pn\_x += SplDiff[0][i + 1] \* w;

}

double f\_x = func(xArr1[k], var);

out << fixed << setprecision(2) << xArr1[k] << " " << setprecision(6) << f\_x << " " << Pn\_x << " ";

out << scientific << abs(f\_x - Pn\_x) << " " << FindErrorForNewton(xArr1[k], xArr, M6);

out << endl;

out.unsetf(ios::scientific);

}

for (int i = 0; i < n; i++)

delete[] SplDiff[i];

delete[] SplDiff;

}

//Вычисление функция фи0

double fi0(double tau)

{

return (1 + 2 \* tau) \* pow(1 - tau, 2);

}

//Вычисление функция фи2

double fi1(double tau)

{

return tau \* pow(1 - tau, 2);

}

// Интерполяция кубическим сплайном дефекта 1

void CubicSpline(double\* xArr1, double\* xArr2, int var)

{

double\* m = new double[n];

//вычисление df0 и dfn

double df0 = Derivative(1, var), dfn = Derivative(2, var);

m[0] = df0;

m[n - 1] = dfn;

double\* alpha = new double[n];

double\* beta = new double[n];

alpha[1] = 0;

beta[1] = df0;

// нахождение параметра m методом прогонки

for (int j = 1; j < n - 1; j++)

{

alpha[j + 1] = -1 / (4 + alpha[j]);

beta[j + 1] = (3 \* (func(xArr1[j + 1], var) - func(xArr1[j - 1], var)) / h - beta[j]) / (4 + alpha[j]);

}

for (int j = n - 2; j >= 0; j--)

m[j] = alpha[j + 1] \* m[j + 1] + beta[j + 1];

// вычисление наибольшего значения 4-й и 5-й производной

double M4 = 0, M5 = 0;

if (var == 1)

{

M4 = pow(2, 2) \* pow(log(2), 4);

M5 = pow(2, 2) \* pow(log(2), 5);

}

else

if (var == 23)

{

M4 = pow(3, 2) \* pow(log(3), 4);

M5 = pow(3, 2) \* pow(log(3), 5);

}

out << endl << "M5 = " << M5 << endl << endl;

// Оценка погрешности интерполяции

out << " x[i] " << " df/dx(x[i]) " << " m[i] " << " Delta " << " Оценка " << endl;

for (int i = 0; i < n; i++)

{

double df\_xi = Derivative(xArr1[i], var);

out << fixed << setprecision(2) << xArr1[i] << " " << setprecision(6) << df\_xi << " " << m[i] << " ";

out << scientific << abs(df\_xi - m[i]) << " " << M5 / 60 \* pow(h, 4) << endl;

out.unsetf(ios::scientific);

}

out << "\nM4 = " << M4 << endl << endl;

// Оценка погрешности интерполяции значения самой функции

out << endl << " x " << " f(x) " << " S31(f;x) " << " Abs(f(x)-S31(f;x)) " << " Оценка " << endl;

for (int i = 0; i < n - 1; i++)

{

double tau = 0.5;

double S\_3\_1 = fi0(tau) \* func(xArr1[i], var) + fi0(1 - tau) \* func(xArr1[i + 1], var) +

h \* (fi1(tau) \* m[i] - fi1(1 - tau) \* m[i + 1]);

double f\_x = func(xArr2[i], var);

out << fixed << setprecision(2) << xArr2[i] << " " << setprecision(6) << f\_x << " " << S\_3\_1 << " ";

out << scientific << abs(f\_x - S\_3\_1) << " " << (M4 / 384 + M5 \* h / 240) \* pow(h, 4) << endl;

out.unsetf(ios::scientific);

}

delete[] alpha;

delete[] beta;

delete[] m;

}

// Вывод матрицы

void PrintMatrix(double\*\* A)

{

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

out << fixed << setprecision(4) << setw(10) << A[i][j];

out << endl;

}

out << endl;

}

// Вывод вектора

void PrintVector(double\* b)

{

for (int i = 0; i < 3; i++)

out << setprecision(16) << b[i] << " ";

out << endl;

}

// Определитель

double Determinant(double\*\* A)

{

return A[0][0] \* A[1][1] \* A[2][2] +

A[0][1] \* A[1][2] \* A[2][0] +

A[0][2] \* A[1][0] \* A[2][1] -

A[0][2] \* A[1][1] \* A[2][0] -

A[0][0] \* A[2][1] \* A[1][2] -

A[2][2] \* A[1][0] \* A[0][1];

}

// Замена столбца матрицы на вектор

void ReplaceColumn(double\*\* A, int j, double\* b)

{

for (int i = 0; i < 3; i++)

A[i][j] = b[i];

}

// Копирование столбца матрицы

void CopyColumn(double\*\* A, int j, double\* b)

{

for (int i = 0; i < 3; i++)

b[i] = A[i][j];

}

// Решение СЛАУ методом Крамера

void CramerMethod(double\*\* A, double\* b, double\* x)

{

double det = Determinant(A);

double\* aj = new double[3];

CopyColumn(A, 0, aj);

ReplaceColumn(A, 0, b);

double det1 = Determinant(A);

ReplaceColumn(A, 0, aj);

CopyColumn(A, 1, aj);

ReplaceColumn(A, 1, b);

double det2 = Determinant(A);

ReplaceColumn(A, 1, aj);

CopyColumn(A, 2, aj);

ReplaceColumn(A, 2, b);

double det3 = Determinant(A);

x[0] = det1 / det;

x[1] = det2 / det;

x[2] = det3 / det;

}

// Функция, получаемая методом среднеквадратичного приближения

double Polinom\_2(double x, double\* c)

{

return c[0] + c[1] \* x + c[2] \* pow(x, 2);

}

//Вычисления погрешности для полинома 2-го порядка

void Error\_2(double\* c, double\* x, double\* x1, int var)

{

for (int i = 0; i < n; i++)

{

out << setprecision(2) << x[i] << " " << setprecision(16) << func(x1[i], var) - Polinom\_2(x[i], c) << endl;

}

}

//Вычисления погрешности для полинома 1-го порядка

void Error\_1(double a0, double a1, double\* x1, int var)

{

double x = 0;

for (int i = 0; i < n; i++)

{

x = 1.0 + i \* 0.2;

out << setprecision(2) << x << " " << setprecision(16) << func(x1[i], var) - a0 - a1 \* x << endl;

}

}

// Среднеквадратичное приближение (дискретный вариант)

void ApproximationDiscrete(double\* xArr1, double\* xArr2, int var)

{

out << "Дискретный вариант\n\n";

//Вычисление (gi,gj)

double\*\* A = new double\* [3];

for (int i = 0; i < 3; i++)

A[i] = new double[3];

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

A[i][j] = 0;

for (int i = 0; i < n; i++)

{

A[0][0] ++;

A[0][1] += xArr1[i];

A[0][2] += pow(xArr1[i], 2);

A[1][2] += pow(xArr1[i], 3);

A[2][2] += pow(xArr1[i], 4);

}

A[1][0] = A[0][1];

A[1][1] = A[0][2];

A[2][0] = A[0][2];

A[2][1] = A[1][2];

out << "Матрица\n";

PrintMatrix(A);

//вычисление вектора правых частей как (f,gj)

double\* b = new double[3];

for (int i = 0; i < 3; i++)

b[i] = 0;

for (int i = 0; i < n; i++)

{

b[0] += func(xArr1[i], var);

b[1] += func(xArr1[i], var) \* xArr1[i];

b[2] += func(xArr1[i], var) \* pow(xArr1[i], 2);

}

out << "\nВектор правых частей\n";

PrintVector(b);

out << endl;

//вычисление коэффициентов сi с помщью метода Крамера

double\* c = new double[3];

CramerMethod(A, b, c);

out << setprecision(5) << "P2(x) = " << c[0] << " + (" << c[1] << ")\*x + (" << c[2] << ")\*x^2\n";

//вычисление нормы погрешности

double scalarFF = 0; // (f,f)

double scalarFG = 0; // (f,gi)

for (int i = 0; i < n; i++)

{

scalarFF += pow(func(xArr1[i], var), 2);

scalarFG += pow(Polinom\_2(xArr1[i], c), 2);

}

double error = sqrt(scalarFF - scalarFG);

out << "Норма погрешности: " << error << endl;

//вычисление погрешности

out << "x " << " Погрешность" << endl;

double\* x = new double[n];

for (int i = 0; i < n; i++)

x[i] = 1.0 + 0.2 \* i;

Error\_2(c, x, xArr1, var);

for (int i = 0; i < 3; i++)

delete[] A[i];

delete[] A;

delete[] b;

delete[] c;

}

// Среднеквадратичное приближение (непрерывный вариант)

void ApproximationContinuous(double\* xArr1, double\* xArr2, int var)

{

out << "\nНепрерывный вариант\n\n";

//Вычисление (gi,gj)

double\*\* A = new double\* [3];

for (int i = 0; i < 3; i++)

A[i] = new double[3];

A[0][0] = 1;

A[0][1] = 3 / 2.;

A[0][2] = 7 / 3.;

A[1][0] = A[0][1];

A[1][1] = A[0][2];

A[1][2] = 15 / 4.;

A[2][0] = A[0][2];

A[2][1] = A[1][2];

A[2][2] = 31 / 5.;

out << "Матрица\n";

PrintMatrix(A);

//Вычисление правых частей (f,gj)

double\* IntegrFG = new double[3];

if (var == 1)

{

IntegrFG[0] = 7.38539008177793;

IntegrFG[1] = 11.6600989499892;

IntegrFG[2] = 18.9824256290058;

}

else

if (var == 23)

{

IntegrFG[0] = 5.46143535976103;

IntegrFG[1] = 8.84904236792789;

IntegrFG[2] = 14.7318165914863;

}

out << "\nВектор правых частей\n";

PrintVector(IntegrFG);

//вычисление сi с помощью метода Крамера

double\* c = new double[3];

CramerMethod(A, IntegrFG, c);

out << setprecision(5) << "P2(x) = " << c[0] << " + (" << c[1] << ")\*x + (" << c[2] << ")\*x^2\n" << endl;

//вычисление нормы погрешности

double IntegrFF\_GG = 0;

if (var == 1)

{

IntegrFF\_GG = 8.9372636945011E-6;

}

else if (var == 23)

{

IntegrFF\_GG = 4.8930208721709E-4;

}

out << "\nНорма погрешности: " << sqrt(IntegrFF\_GG) << endl << endl;

//вычисление погрешности

out << "x " << " Погрешность" << endl;

double\* x = new double[n];

for (int i = 0; i < n; i++)

x[i] = 1.0 + 0.2 \* i;

Error\_2(c, x, xArr1, var);

for (int i = 0; i < 3; i++)

delete[] A[i];

delete[] A;

delete[] IntegrFG;

delete[] c;

}

//Поиск функции L

double lossL(double x, int var)

{

return func(x, var) - a0 - a1 \* x;

}

//Метод половинного деления

double find\_d(int var, double a1, double b1, double a2)

{

double x = (a1 + b1) / 2;

while (abs(Derivative(x, var) - a2) > eps)

{

if (Derivative(x, var) - a2 > 0)

b1 = x;

else

a1 = x;

x = (a1 + b1) / 2;

}

return x;

}

//Вычисление значений для равномерного приближения

void UniformApproximation(double a, double b, int var)

{

a1 = (func(b, var) - func(a, var)) / (b - a);

d = find\_d(var, a, b, a1);

a0 = (func(a, var) + func(d, var) - a1 \* (a + d)) / 2;

aL = lossL(a, var);

bL = lossL(b, var);

dL = lossL(d, var);

}

// Метод обратной интерполяции

void ReverseInterpolation(double\* xArr, int var)

{

// Значение функции в середине отрезка

double c = (func(a, var) + func(b, var)) / 2;

double\* y = new double[n];

//y-аргумент, х - функция

for (int i = 0; i < n; i++)

{

y[i] = func(xArr[i], var) - c;

}

// Таблица разделённых разностей

double\*\* SplDiff = new double\* [n];

for (int i = 0; i < n; i++)

SplDiff[i] = new double[n + 1];

FindDividedDifferenceMatrix(SplDiff, y, xArr);

PrintSplitDifferenceMatrix2(SplDiff);

//Формула Ньютона, для нахождения корня P\_n(0)

double x = SplDiff[0][1];

double w = 1;

for (int i = 1; i < n; i++)

{

w \*= -y[i - 1];

x += SplDiff[0][i + 1] \* w;

}

out << "c = " << c << endl;

out << endl << "Корень " << x << endl;

out << "Невязка = Abs(f(x) - c) = " << setprecision(12) << abs(func(x, var) - c) << endl;

for (int i = 0; i < n; i++)

delete[] SplDiff[i];

delete[] SplDiff;

delete[] y;

}

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int var;

cout << "Введите номер варианта" << endl;

cout << "Вариант № ";

cin >> var;

out << "Вариант № " << var << endl;

double\* xArr = new double[n];

double\* xArr1 = new double[n];

out << "Интерполяционная формула Ньютона" << endl;

Newton(xArr, xArr1, yArr, var);

out << endl << "Интерполяция кубическим сплайном" << endl;

CubicSpline(xArr, xArr1, var);

out << endl << "Среднеквадратичное приближение" << endl << endl;

ApproximationDiscrete(xArr, xArr1, var);

ApproximationContinuous(xArr, xArr1, var);

out << endl << endl << "Равномерное приближение" << endl;

UniformApproximation(a, b, var);

out << "P1(x)= " << setprecision(5) << a0 << " + " << a1 << "\*x" << ", d = " << d << endl;

out << "L(a)=" << aL << ", L(d)=" << dL << ", L(b)=" << bL << endl << endl;;

out << "x " << "Погрешность" << endl;

Error\_1(a0, a1, xArr, var);

out << endl << "Решение уравнения методом обратной интерполяции" << endl << endl;

ReverseInterpolation(xArr, var);

delete[] xArr;

delete[] xArr1;

delete[] yArr;

system("pause");

}