Федеральное государственное образовательное бюджетное учреждение высшего профессионального образования

Финансовый Университет при Правительстве РФ

Колледж информатики и программирования

Практическая работа №

По дисциплине: Численные методы программирования

Выполнил студент

Группы 3ПКС-416

Ермаков И.Н.

Москва 2018

Код программы:

**Метод Тейлора**.Предполагая, что точное решение http://konspekta.net/megaobuchalkaru/imgbaza/baza6/1785239492096.files/image018.png задачи (1) является аналитической функцией в некоторой окрестности http://konspekta.net/megaobuchalkaru/imgbaza/baza6/1785239492096.files/image037.png точки http://konspekta.net/megaobuchalkaru/imgbaza/baza6/1785239492096.files/image039.png , разложим http://konspekta.net/megaobuchalkaru/imgbaza/baza6/1785239492096.files/image018.png в ряд Тейлора в точке http://konspekta.net/megaobuchalkaru/imgbaza/baza6/1785239492096.files/image041.png

#include "pch.h"

#include <iostream>

#include <iomanip>

#include <cmath>

using namespace std;

double sinx(double x, double e);

int main()

{

double x, e;

cout << "Enter x: ";

cin >> x;

cout << "Enter e: ";

cin >> e;

cout << "sin(x^2/2) = " << sinx(x, e) << endl;

return 0;

}

double sinx(double x, double e)

{

double s = x \* x / 2;

double sum = 0.0;

int i = 0;

do

{

sum += s;

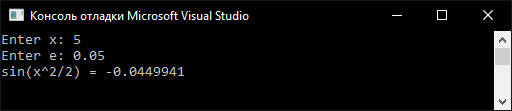
i++;

s \*= (-1.0 \* x \* x \* x \* x / 4) / (2 \* i \* (2 \* i + 1));

} while (fabs(s) > e);

return sum;

}



Код программы:

**Интерполяцио́нный многочле́н Лагра́нжа** — многочлен минимальной степени, принимающий данные значения в данном наборе точек. Для *n+1* пар чисел (*x0*, *y0*), (*x1*, *y1*),…, (*xn*, *yn*), где все *xj* различны, существует единственный многочлен *L(x)* степени не более *n*, для которого *L(xj) = yj*.

#include "pch.h"

#include "iostream"

#include "math.h"

#include "stdlib.h"

#include "locale.h"

#include "conio.h"

using namespace std;

int main()

{

setlocale(LC\_ALL, "rus");

const int n = 6;

double X[n] = { 0,0.2,0.4,0.6,0.8,1 }, Y[n] = { 0.75,1.1,1.35,1.25,1.05,0.8 };

double x = 0.96;

double L = 0;

double F = 1;

int i, j;

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

{

F = 1;

for (j = 0; j < n; j++)

{

if (j != i)

{

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

}

}

F = F \* Y[i];

L = L + F;

}

cout<<"\nЗначение в т.Х="<<L;

return 0;

}



Код программы:

Ньютон(разделение разности) - Первая и вторая формулы Ньютона предполагают, что узлы интерполирования являются равноотстоящими. Однако, в общем случае функция *f*(*x*) может быть задана таблицей, в которой узлы находятся на произвольном расстоянии друг от другаhttps://studfiles.net/html/2706/211/html_rcOSe9y2KF.vKQV/img-DIKJ38.png, где значения*hi*(*i*= https://studfiles.net/html/2706/211/html_rcOSe9y2KF.vKQV/img-VkS7nU.png) являются различными.

#include "pch.h"

#include "iostream"

#include "math.h"

#include "stdlib.h"

#include "locale.h"

#include "conio.h"

int main(void) {

const int n = 5;

double Xi[n] = { 1,2,3,4,5 };

double Yi[n] = { 2,3,4,5,6 };

double X = 0.1;

double f, LN, XXX, XX = 1.;

int i, j, k;

for (i = 1, LN = Yi[0]; i < n; i++)

{

XX \*= (X - Xi[i - 1]);

for (j = 0, f = 0; j <= i; j++)

{

for (k = 0, XXX = 1.; k <= i; k++)

{

if (k != j)

XXX \*= Xi[j] - Xi[k];

}

f += Yi[j] / XXX;

}

LN += XX \* f;

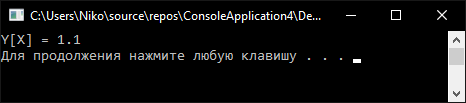
}

printf("Y[X] = %g\n", LN);

system("pause");

return 0;

}



Код программы:

Аппроксимация лагранжа

#include "pch.h"

#include <iostream>

#include <Windows.h>

#include <math.h>

using namespace std;

double TestF(double x);

double InterpolateLagrangePolynomial(double x, double\* xValues, double\* yValues, int size);

int main()

{

setlocale(LC\_ALL, "ru");

const int size = 5;

double\* xValues = new double[size];

double\* yValues = new double[size];

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

{

xValues[i] = i;

yValues[i] = TestF(i);

}

cout << InterpolateLagrangePolynomial(13.6, xValues, yValues, size);

return 0;

}

double TestF(double x)

{

return x \* x\*x + 3 \* x\*x + 3 \* x + 1;

}

double InterpolateLagrangePolynomial(double x, double\* xValues, double\* yValues, int size)

{

double lagrangePol = 0;

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

{

double basicsPol = 1;

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

{

if (j != i)

{

basicsPol \*= (x - xValues[j]) / (xValues[i] - xValues[j]);

}

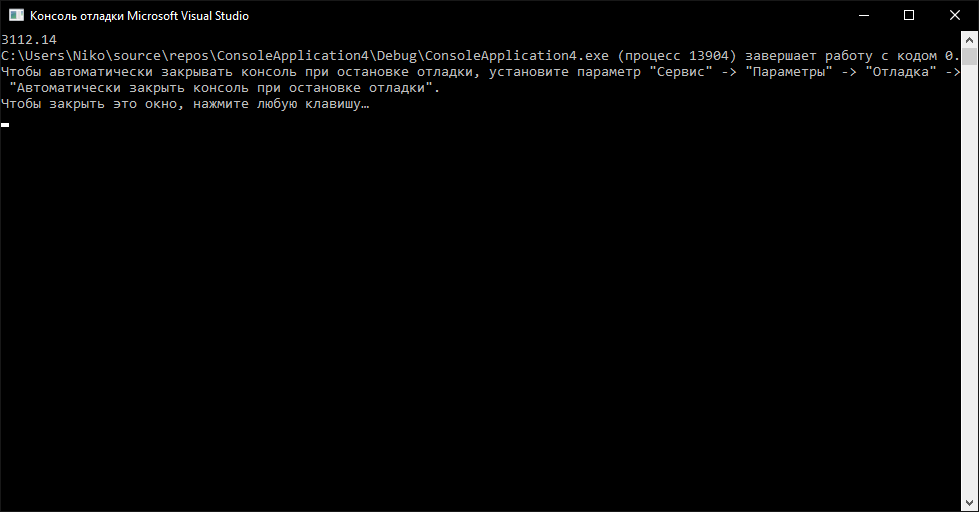
}

lagrangePol += basicsPol \* yValues[i];

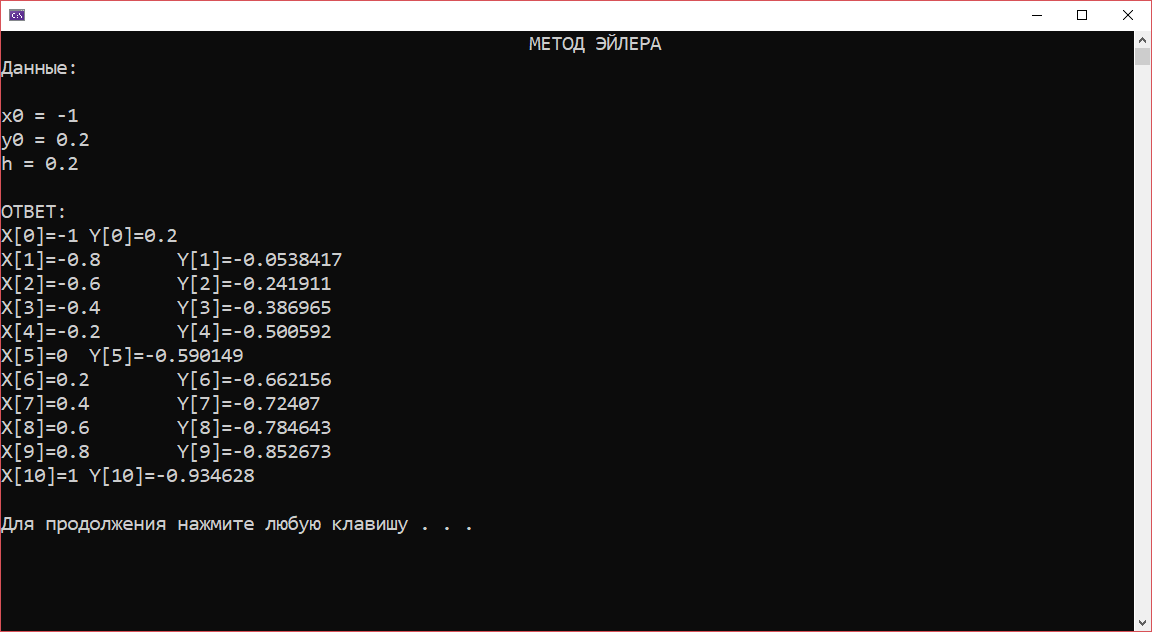
}

return lagrangePol; cout << "\n" << endl; system("pause"); return 0;

}



**Скриншот программы:**



**Листинг:**

//метод Эйлера

//Метод Эйлера являлся исторически первым методом численного решения задачи Коши.

//Ввиду невысокой точности и вычислительной неустойчивости для практического нахождения //решений задачи Коши метод Эйлера применяется редко.

//Однако в виду своей простоты метод Эйлера находит своё применение в теоретических //исследованиях дифференциальных уравнений, задач вариационного исчисления и ряда других //математических проблем. Метод Эйлера является явным, одношаговым методом первого //порядка точности. Он основан на аппроксимации интегральной кривой кусочно-линейной //функцией, так называемой ломаной Эйлера.

#include "pch.h"

#include <iostream>

#include <cmath>

using namespace std;

double F(double x, double y);

int main()

{

setlocale(LC\_ALL, "ru");

system("title ");

double a = -1; double b = 1; double h = 0.2;

double n = (b - a) / h;

double\* X = new double[(int)n];

double\* Y = new double[(int)n];

X[0] = a; Y[0] = 0.2;

cout << "\t\t\t\t\t\tМЕТОД ЭЙЛЕРА" << endl;

cout << "Данные: " << endl << endl;

cout << "x0 = " << X[0] << endl;

cout << "y0 = " << Y[0] << endl;

cout << "h = " << h << endl;

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

{

X[i] = a + i \* h;

Y[i] = Y[i - 1] + h \* F(X[i - 1], Y[i - 1]);

}

cout << endl << "ОТВЕТ:" << endl;

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

{

cout << "X[" << i << "]=" << X[i] << "\tY[" << i << "]=" << Y[i] << endl;

}

cout << endl;

system("pause");

return 0;

}

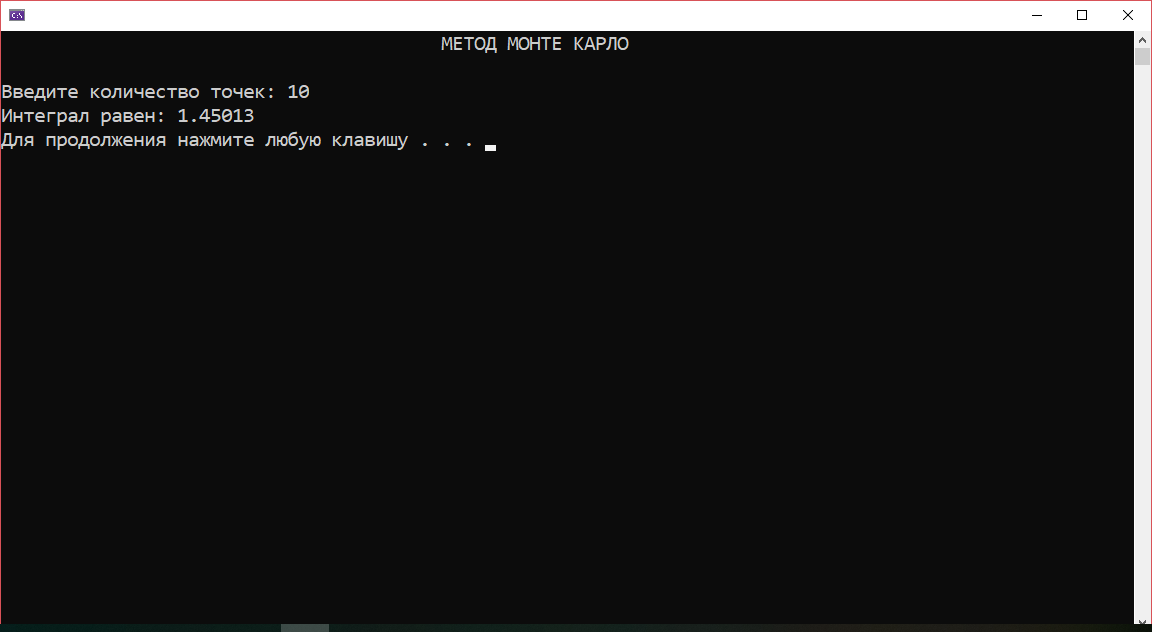
double F(double x, double y)

{

return cos(1.5 \* x - pow(y, 2)) - 1.3;

}

**Скриншот программы:**



**Листинг:**

//Метод Монте Карло

//Ме́тоды Мо́нте-Ка́рло (ММК) — группа численных методов для изучения случайных процессов.

//Суть метода заключается в следующем: процесс моделируется при помощи генератора //случайных величин.

//Это повторяется много раз, а потом на основе полученных случайных данных вычисляются //вероятностные

//характеристики решаемой задачи. Например, чтобы узнать, какое в среднем будет //расстояние между двумя

//случайными точками в круге, методом Монте-Карло, нужно взять много случайных пар точек, //для каждой пары

//найти расстояние, а потом усреднить.

#include "pch.h"

#include <iostream>

#include <stdlib.h>

#include <stdio.h>

#include <time.h>

double func(double x)

{

return 1 - x \* x;

}

using namespace std;

int main()

{

setlocale(LC\_ALL, "ru");

system("title ");

int point\_a = -1;

int point\_b = 1;

int number\_of\_random = 0;

double s = 0;

cout << "\t\t\t\t\tМЕТОД МОНТЕ КАРЛО" << endl << endl;

cout << "Введите количество точек: ";

cin >> number\_of\_random;

srand((unsigned)time(NULL));

for (int i = 0; i < number\_of\_random; i++)

{

s += func(point\_a + ((double)rand() / RAND\_MAX \* (point\_b - point\_a)));

}

s = s / (double)number\_of\_random\*(point\_b - point\_a);

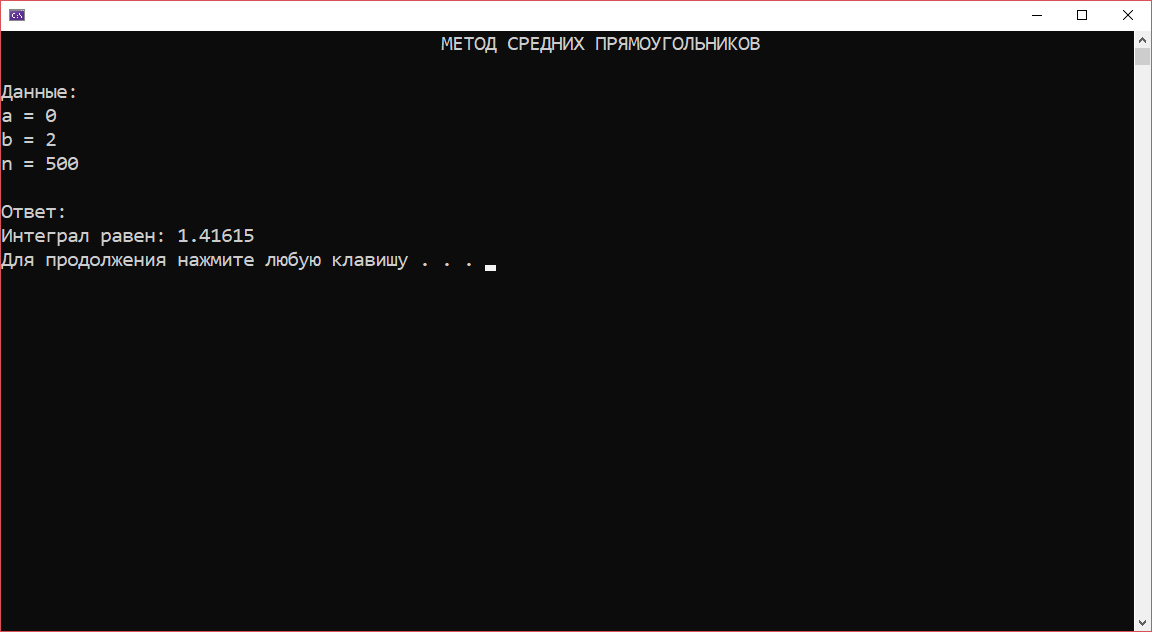
cout << "Интеграл равен: " << s << endl;

system("pause");

return 0;

}

**Скриншот программы:**



**Листинг:**

//метод средних прямоугольников

//Метод прямоугольников — метод численного интегрирования функции одной переменной, //заключающийся в замене подынтегральной функции на многочлен нулевой степени, то есть //константу, на каждом элементарном отрезке. Если рассмотреть график подынтегральной //функции, то метод будет заключаться в приближённом вычислении

//площади под графиком суммированием площадей конечного числа прямоугольников, ширина //которых будет определяться расстоянием между соответствующими соседними узлами //интегрирования, а высота — значением подынтегральной функции

//в этих узлах. Алгебраический порядок точности равен 0. (Для формулы средних //прямоугольников равен 1).

#include "pch.h"

#include <iostream>

#include <Windows.h>

using namespace std;

double InFunction(double x);

double CalcIntegral(double a, double b, int n);

int main(void)

{

setlocale(LC\_ALL, "ru");

system("title ");

double integral;

double a = 0, b = 2;

int n = 500;

cout << "\t\t\t\t\tМЕТОД СРЕДНИХ ПРЯМОУГОЛЬНИКОВ" << endl << endl;

cout << "Данные: " << endl;

cout << "a = " << a << endl;

cout << "b = " << b << endl;

cout << "n = " << n << endl;

integral = CalcIntegral(a, b, n);

cout << endl << "Ответ:" << endl;

cout << "Интеграл равен: " << integral << endl;

system("pause");

return 0;

}

double InFunction(double x) //Подынтегральная функция

{

return sin(x); //Например, sin(x)

}

double CalcIntegral(double a, double b, int n)

{

int i;

double result, h;

result = 0;

h = (b - a) / n; //Шаг сетки

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

{

result += InFunction(a + h \* (i + 0.5)); //Вычисляем в средней точке и добавляем в сумму

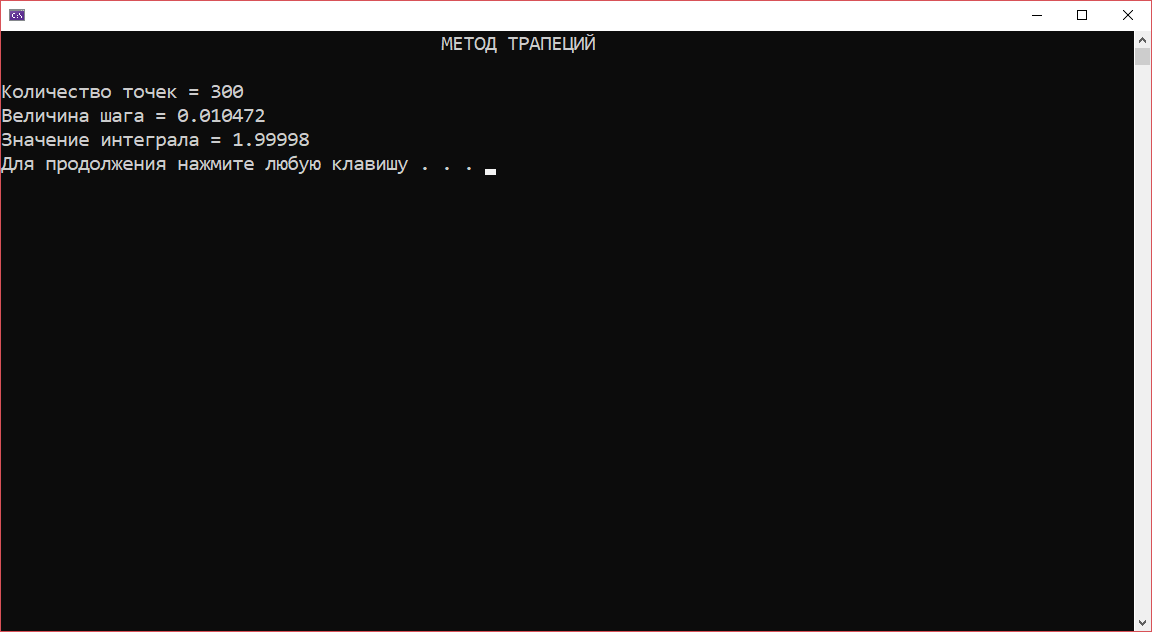
}

result \*= h;

return result;

}

**Скриншот программы:**



**Листинг:**

//метод трапеций

//Метод трапеций — метод численного интегрирования функции одной переменной, //заключающийся в замене на каждом элементарном отрезке подынтегральной функции на //многочлен первой степени, то есть линейную функцию.

//Площадь под графиком функции аппроксимируется прямоугольными трапециями. Алгебраический //порядок точности равен 1.

#include "pch.h"

#include <iostream>

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#define NUMPOINT 300

#define PI 3.1415926536

using namespace std;

double Integral(double \*f, double step);

int main()

{

setlocale(LC\_ALL, "ru");

system("title ");

double \*f, step, t, S;

int i;

f = (double\*)malloc(NUMPOINT \* sizeof(double));

cout << "\t\t\t\t\tМЕТОД ТРАПЕЦИЙ" << endl << endl;

cout << "Количество точек = " << NUMPOINT << endl;

step = PI / NUMPOINT; // величина шага (высота трапеций)

cout << "Величина шага = " << step << endl;

t = 0.0;

// Инициализация значений функции f(t)=sin(t)

for (i = 0; i < NUMPOINT; i++)

{

f[i] = sin(t);

t += step;

}

S = Integral(f, step); // вычисление интеграла

cout << "Значение интеграла = " << S << endl;

system("pause");

return 0;

}

// Функция вычисления определенного интеграла

double Integral(double \*f, double step)

{

double value = 0;

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

{

value += f[i];

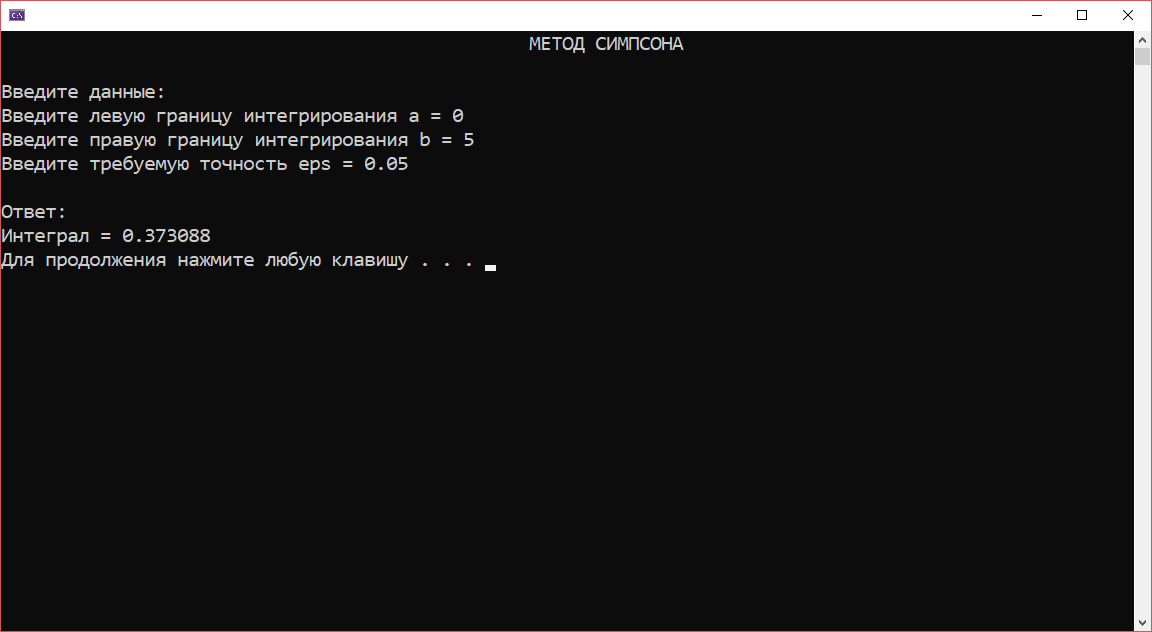
}

value \*= step;

return value;

}

**Скриншот программы:**



**Листинг:**

//метод Симпсона

//Суть метода заключается в приближении подынтегральной функции на отрезке {\displaystyle //[a,b]} [a,b]

//интерполяционным многочленом второй степени {\displaystyle p\_{2}(x)} p\_{2}(x),

//то есть приближение графика функции на отрезке параболой. Метод Симпсона имеет порядок //погрешности 4 и алгебраический порядок точности 3.

#include "pch.h"

#include <iostream>

#include <math.h>

#include <cmath>

using namespace std;

typedef double(\*pointFunc)(double);

double f(double x);

double simpson\_integral(pointFunc f, double a, double b, int n);

int main()

{

setlocale(LC\_ALL, "ru");

system("title ");

double a, b, eps;

double s1, s;

int n = 1; //начальное число шагов

cout << "\t\t\t\t\t\tМЕТОД СИМПСОНА" << endl << endl;

cout << "Введите данные:" << endl;

cout << "Введите левую границу интегрирования a = ";

cin >> a;

cout << "Введите правую границу интегрирования b = ";

cin >> b;

cout << "Введите требуемую точность eps = ";

cin >> eps;

s1 = simpson\_integral(f, a, b, n); //первое приближение для интеграла

do

{

s = s1; //второе приближение

n = 2 \* n; //увеличение числа шагов в два раза,

//т.е. уменьшение значения шага в два раза

s1 = simpson\_integral(f, a, b, n);

} while (fabs(s1 - s) > eps); //сравнение приближений с заданной точностью

cout << endl << "Ответ:" << endl;

cout << "Интеграл = " << s1 << endl;

system("pause");

return 0;

}

double f(double x)

{

return x/(pow(x,4) + 4);

}

double simpson\_integral(pointFunc f, double a, double b, int n)

{

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

double k1 = 0, k2 = 0;

for (int i = 1; i < n; i += 2)

{

k1 += f(a + i \* h);

k2 += f(a + (i + 1)\*h);

}

return h / 3 \* (f(a) + 4 \* k1 + 2 \* k2);

}

Ниже представлен код программы:

//метод итераций

#define \_USE\_MATH\_DEFINES

#include <iostream>

#include <cmath>

using namespace std;

double find(double x, double eps)

{

double rez; int iter = 0;

cout << "x0= " << x << " ";

do {

rez = x;

x = 1 / (sin(M\_PI\*x / 180));

iter++;

} while (fabs(rez - x) > eps && iter<20000);

cout << iter << " iterations" << endl;

return x;

}

int main()

{

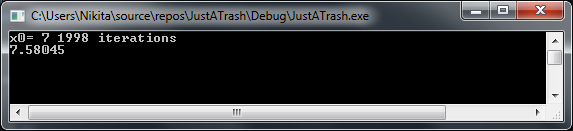
cout << find(7, 0.00001);

cin.get();

return 0;

}

Ниже представлен скриншот выполнения программы:



Ниже представлен код программы:

// метод ньютона

#define \_USE\_MATH\_DEFINES

#include <iostream>

#include <cmath>

using namespace std;

double find(double x, double eps)

{

double f, df; int iter = 0;

cout << "x0= " << x << " ";

do {

f = sin(M\_PI\*x / 180) - 1 / x;

df = M\_PI / 180 \* cos(M\_PI\*x / 180) + 1 / (x\*x);

x = x - f / df;

iter++;

} while (fabs(f) > eps && iter<20000);

cout << iter << " iterations" << endl;

return x;

}

int main()

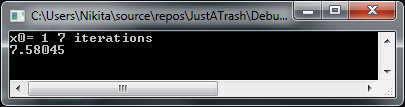
{

cout << find(1, 0.00001);

cin.get(); return 0;

}

Ниже представлен скриншот выполнения программы:



Ниже представлен код программы:

//метод хорд

#define \_USE\_MATH\_DEFINES

#include <iostream>

#include <cmath>

using namespace std;

double find(double x0, double x1, double eps)

{

double rez = x1, f0, f;

int iter = 0;

cout << "x0= " << x0 << " x1= " << x1 << " ";

do {

f = sin(M\_PI\*rez / 180) - 1 / rez;

f0 = sin(M\_PI\*x0 / 180) - 1 / x0;

rez = rez - f / (f - f0)\*(rez - x0);

iter++;

} while (fabs(f) > eps && iter<20000);

cout << iter << " iterations" << endl;

return rez;

}

int main()

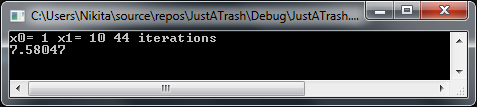
{

cout << find(1.0, 10.0, 0.000001);

cin.get(); return 0;

}

Ниже представлен скриншот выполнения программы:



Ниже представлен код программы:

//метод половинного деления

#define \_USE\_MATH\_DEFINES

#include <iostream>

#include <cmath>

using namespace std;

double func(double x)

{

return (sin(M\_PI\*x / 180) - 1 / x);

}

double find(double x0, double x1, double eps)

{

double left = x0, right = x1, x, fl, fr, f;

int iter = 0;

cout << "x0= " << x0 << " x1= " << x1 << " ";

do {

x = (left + right) / 2;

f = func(x);

if (f > 0) right = x;

else left = x;

iter++;

} while (fabs(f) > eps && iter<20000);

cout << iter << " iterations" << endl;

return x;

}

int main()

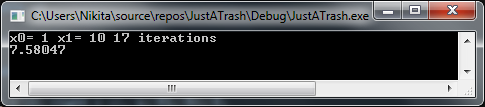
{

cout << find(1.0, 10.0, 0.000001);

cin.get(); return 0;

}

Ниже представлен скриншот выполнения программы:



Ниже представлен код программы:

//метод Зейделя

#include <iostream>

using namespace std;

const int max\_it = 500;

int m\_size;

double \*\*matrix;

double \*\*b;

double \*c;

double \*buff\_x;

double \*x;

double E;

bool guarantee = true;

void PushX()

{

for (int i = 0; i < m\_size; i++)

buff\_x[i] = x[i];

}

void ShowMatrix()

{

cout << endl;

for (int i = 0; i < m\_size; i++)

{

cout << "( ";

for (int j = 0; j < m\_size; j++)

{

cout.precision(4);

cout << matrix[i][j] << " ";

}

cout << "|" << matrix[i][m\_size];

cout << " )" << endl;

}

cout << endl;

}

int main()

{

setlocale(LC\_ALL, "rus");

cout << "Введите размерность матрицы: ";

cin >> m\_size;

cout << "Задайте элементы расширенной матрицы, разделяя элементы пробелами" << endl;

matrix = new double\*[m\_size];

b = new double\*[m\_size];

c = new double[m\_size];

x = new double[m\_size];

buff\_x = new double[m\_size];

for (int i = 0; i < m\_size; i++)

{

matrix[i] = new double[m\_size + 1];

b[i] = new double[m\_size];

for (int j = 0; j < m\_size + 1; j++)

cin >> matrix[i][j];

}

cout << "Введите Эпсилон: ";

cin >> E;

cout << endl << "Введенная матрица: ";

ShowMatrix();

cout << "E = " << E << endl;

//Проверка на сходимость

for (int ij = 0; ij < m\_size; ij++)

{

double val = abs(matrix[ij][ij]);

double sum = 0;

for (int j = 0; j < m\_size; j++)

if (ij != j)

sum += abs(matrix[ij][j]);

if (val <= sum)

{

guarantee = false;

break;

}

}

cout << "Гарантия сходимости: " <<((guarantee) ? "Да" : "Нет") << endl;

//Заполнение матрицы b и вектора c

for (int i = 0; i < m\_size; i++)

{

c[i] = matrix[i][m\_size] / matrix[i][i];

x[i] = c[i];

for (int j = 0; j < m\_size; j++)

if (i == j)

b[i][j] = 0;

else

b[i][j] = -(matrix[i][j] / matrix[i][i]);

}

//Основной цикл (итерации)

for (int it = 0; it < max\_it; it++)

{

PushX();

//Вычисление xi

for (int i = 0; i < m\_size; i++)

{

//xi = ci+(bij\*xj)

x[i] = c[i];

for (int j = 0; j < m\_size; j++)

{

if (i == j) continue;

x[i] += b[i][j] \* buff\_x[j];

}

}

}

cout << "Ответы: " << endl;

for (int i = 0; i < m\_size; i++)

{

cout << "x" << i + 1 << " = " << x[i] << endl;

}

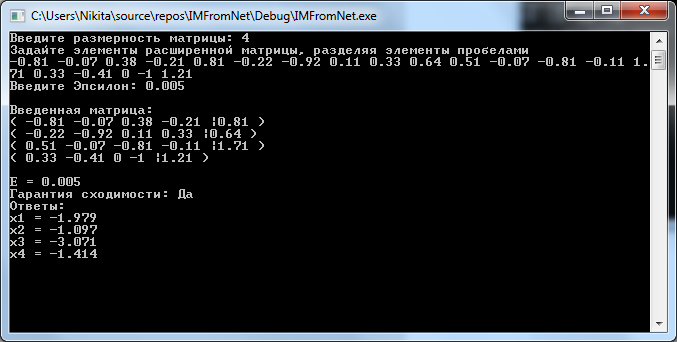
getchar();

getchar();

return 0;

}

Ниже представлен скриншот выполнения программы:



Ниже представлен код программы:

//комбинированный метод

#include <iostream>

#include <math.h>

#include <conio.h>

using namespace std;

float f(float x)

{

return (1.0 / (1.05\*x)) - pow(x, 2);

}

float fp(float x)

{

return -1 / pow(sin(x), 2) - 2 \* x;

}

float horda(float a, float b, float eps)

{

int ii = 0;

float c;

do

{

c = a - f(a)\*(b - a) / (f(b) - f(a));

if (f(a)\*f(c)<0)

{

b = c;

}

else

{

a = c;

}

ii++;

} while (fabs(f(c))>eps);

cout<<"iteraciy: "<<ii<<endl;

return c;

}

float Newton(float a, float eps)

{

int ii = 0;

do

{

a = a - f(a) / fp(a);

ii++;

} while (fabs(f(a))> eps);

cout<<"iteraciy: "<<ii<<endl;

return a;

}

void main()

{

//clrscr();

float a, b, eps = 0.001;

cout<<"a = "; cin>>a;

cout<<"b = "; cin>>b;

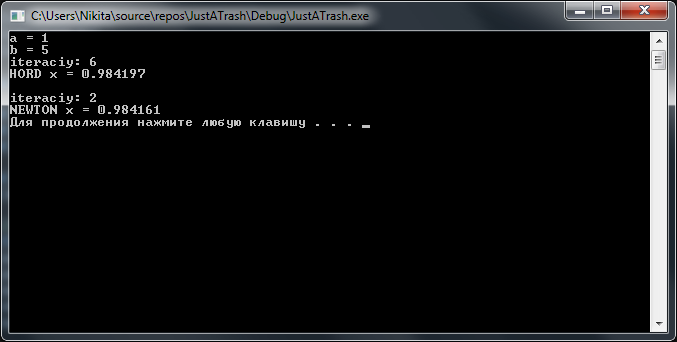
cout<<"HORD x = "<<horda(a, b, eps)<<"\n\n";

cout<<"NEWTON x = "<<Newton(a, eps)<<"\n";

system("pause");

}

Ниже представлен скриншот выполнения программы:



Ниже представлен код программы:

//апроксимация лагранжа

#include <iostream>

#include <math.h>

#include <conio.h>

using namespace std;

double InterpolateLagrangePolynomial(double x, double\* xValues, double\* yValues, int size);

double TestF(double x);

int main(void) {

const int size = 10;

double xValues[size];

double yValues[size];

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

{

xValues[i] = i;

yValues[i] = TestF(i);

}

cout << InterpolateLagrangePolynomial(13.6, xValues, yValues, size) << "\n";

system("pause");

}

double InterpolateLagrangePolynomial(double x, double\* xValues, double\* yValues, int size)

{

double lagrangePol = 0;

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

{

double basicsPol = 1;

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

{

if (j != i)

{ basicsPol \*= (x - xValues[j]) / (xValues[i] - xValues[j]); }

}

lagrangePol += basicsPol \* yValues[i];

}

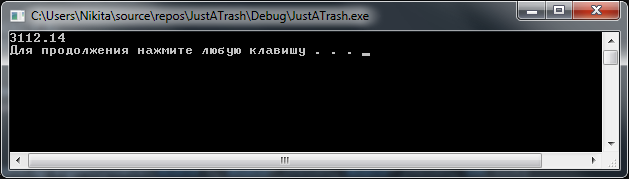
return lagrangePol;

}

double TestF(double x)

{ return x \* x\*x + 3 \* x\*x + 3 \* x + 1; // for example }

Ниже представлен скриншот выполнения программы:



Ниже представлен код программы:

//метод тейлора

#include <iostream>

#include <iomanip>

#include <cmath>

using namespace std;

double sinx(double x, double e);

int main()

{

double x, e;

cout << "Enter x: ";

cin >> x;

cout << "Enter e: ";

cin >> e;

cout << "sin(x^2/2) = " << sinx(x, e) << endl;

system("pause");

return 0;

}

double sinx(double x, double e)

{

double s = x \* x / 2;

double sum = 0.0;

int i = 0;

do

{

sum += s;

i++;

s \*= (-1.0 \* x \* x \* x \* x / 4) / (2 \* i \* (2 \* i + 1));

} while (fabs(s) > e);

return sum;

}

Ниже представлен скриншот выполнения программы:

