**Задание №4. Вычисление определенного интеграла**

**Цель задания:** практическое освоение методов управления численным процессом с целью обеспечения заданной точности вычисления определенного интеграла за приемлемое время, используя составные квадратурные формулы с методами оценки погрешности и выбора оптимального разбиения.

Вариант №8

***Часть №1: Квадратурные формулы Ньютона-Кот(е)са и Гаусса***

* 1. Реализовать методы вычисления определенного интеграла с использованием составных квадратурных формул: *средних* и *левых прямоугольников*, *трапеции*, *Симпсона.* В качестве подынтегральной функции взять функцию Вашего варианта (т.е. положить весовую функцию :
  2. Реализовать методы вычисления определенного интеграла с использованием составных квадратурных формул на базе 3-х-точечных формул Ньютона-Кот(е)са и Гаусса.

В качестве подынтегральной функции взять функцию *,* где *f(x)* и  *—* функции в Вашем варианте задания:

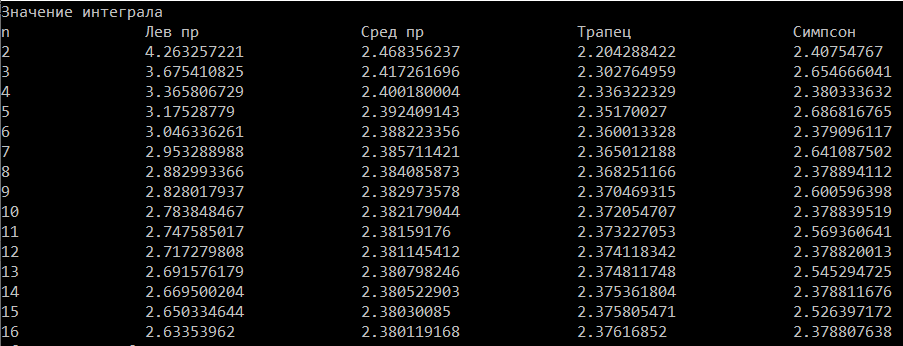
***Примечание***: узлы каждой малой 3-х-точечной квадратурной формулы Гаусса находить с помощью формул Кардано.

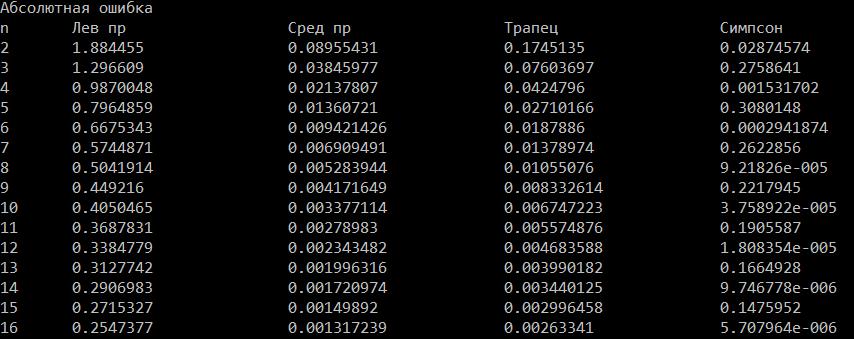
* 1. Для каждой квадратурной формулы из пп.1.1-2 представить график зависимости абсолютной погрешности от количества разбиений интервала интегрирования.

Полный текст программы на языке С++ приведён в приложении.

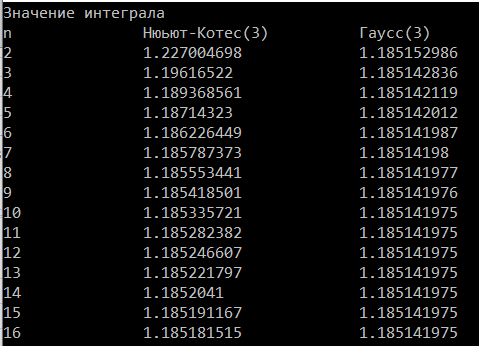
Точные значения интегралов при с 16-ю знаками после запятой:

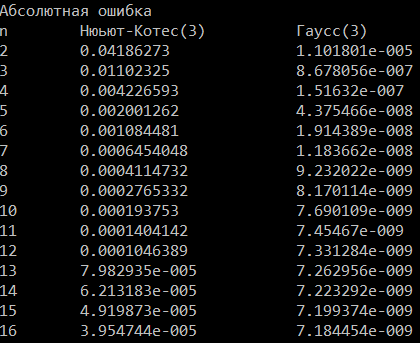
Результаты расчёта программы интеграла и абсолютные погрешности:





Примечание: для метода Симпсона число разбиений отрезка должно быть чётным, поэтому значения для нечётных n можно не учитывать.





Графики зависимости абсолютной погрешности от количества разбиений интервала интегрирования:

Для методов Симпсона, Гаусса и Ньютона-Котеса использовали логарифмическую ось ординат, поскольку погрешность очень быстро убывает с ростом числа отрезков разбиения.

Таким образом, метод Гаусса имеет наибольшую скорость сходимости – порядок сходимости метода равен примерно 2n – 1 = 5.

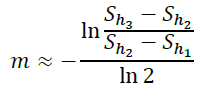
Заметим, что с ростом числа разбиений отрезка будет накапливаться вычислительная ошибка, связанная с решением СЛАУ при определении коэффициентов квадратурных формул, а также вычислений самих узлов для метода Гаусса.

***Часть №2: Методы оценки погрешности составных квадратурных формул***

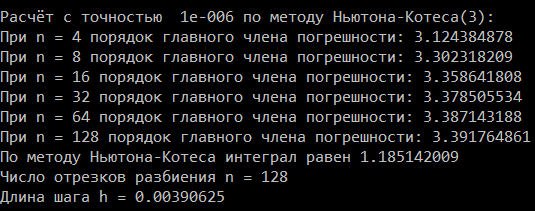
* 1. Вычислить определенный интеграл с заданной точностью c использованием составной 3-х-точечной квадратурной *формулы Ньютона-Кот(е)са*. Погрешность оценивать *методом Ричардсона*. На каждых последовательных трех точках оценивать скорость сходимости по *правилу Эйткена*. Указать длину шага *h* разбиения интервала интегрирования, при котором была достигнута требуемая точность *.*
  2. Выполнить всё, что указано в п. 2.1, используя 3-х-точечные *формулы Гаусса* вместо формул Ньютона-Кот(е)са.
  3. Проведя вычисления (для одной из построенных квадратурных формул из п.2.1-2) по трем сеткам с малым числом шагов (например, 1, 2 и 4) и используя оценку скорости сходимости по Эйткену, выбрать шаг *hopt.* Начать расчет с полученного шага и снова довести вычисления интеграла до требуемой точности *.* Указать шаг разбиения интервала интегрирования, при котором достигнута требуемая точность, и сравнить его с шагом, вычисленным в 2.1 или 2.2 (в зависимости от того, какую квадратурную формулу Вы выбрали для выполнения задания в п. 2.3).

**Выполнение**

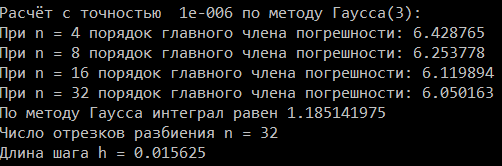
Вычисляем с точностью , начинаем с n = 1 разбиения отрезка, последовательно уменьшая шаг вдвое. Погрешность оцениваем методом Ричардсона, т.е. на каждом шаге решаем систему, из которой находим уточнённое значение интеграла и коэффициенты разложения погрешности по степеням . На каждых последовательных трех точках оцениваем скорость сходимости по правилу Эйткена.



Результаты для метода Ньютона-Котеса (для метода Ричардсона полагаем ):



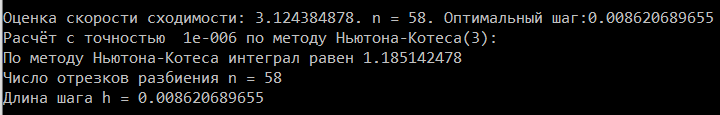
Результаты для метода Гаусса (для метода Ричардсона полагаем ):



Таким образом, в методе Гаусса по трём точкам скорость сходимости выше, поскольку узлы разбиения подбираются специальным образом. По правилу Эйткена оценка скорости сходимости выше предполагаемых значений 3 и 5 соответственно для метода Ньютона-Котеса и Гаусса.

Найдём оптимальный шаг для метода Ньютона-Котеса с помощью расчёта для 1,2 и 4 разбиения.

По оптимальному шагу находим оптимальное число точек разбиения (округлив вверх до целого). Оптимальный шаг также корректируем, исходя из целого n.



Получили оптимальное число разбиений n = 58 и оптимальный шаг . Начинаем расчёт по методу Ньютона-Котеса с числа точек n/2 = 28, чтобы можно было оценить по Ричардсону погрешность при n = 58. В результате оценка погрешности по Ричардсону при n = 58 действительно не превосходит .

Таким образом, нахождение оптимального шага позволяет не рассчитывать интегралы при заведомо недостаточных числах точках разбиения. Кроме того, по сравнению с шагами, являющимися степенями 2 достигнута экономия – вместо n = 128 достаточно n = 58 точек разбиения.

В данном случае оценка m по правилу Эйткена для метода Ньютона-Котеса немного растёт с ростом n, поэтому найденного оказалось достаточно для вычисления с точностью .

Приложение

Листинг программы

#include <iostream>

#include <math.h>

#include <iomanip>

#include <conio.h>

#include <locale>

#define N 2

#define N1 16

using namespace std;

const double PI = 3.1415926535897932384626433832795;

const double Itocn = 2.3788019296148555;

const double ItocnP = 1.185141967803177;

double a = 1.8, b = 2.3, alpha = 0, beta = 0.6;

double f(double x)

{

return 3.7\*cos(1.5\*x)\*exp(-4.0\*x/3.0)+2.4\*sin(4.5\*x)\*exp(2.0\*x/3.0)+4.0;

}

double p(double x)

{

return pow(x-a,-alpha)\*pow(b-x,-beta);

}

int sign (double n)

{

while (n < 0)

return -1;

while (n == 0)

return 0;

return 1;

}

//Решение системы классическим методом Гаусса

double\* GaussSys(int NN, double (\*A)[N+1],double B[])

{

int nx=NN, i, j, k;

double d, s;

double \*X = new double[nx];//столбец с решением

for (k = 0; k < nx; k++) // прямой ход

{

for (j = k + 1; j < nx; j++)

{

if (A[j][k]!=0)

{

d = A[j][k] / A[k][k];

for (i = k; i < nx; i++)

{

A[j][i] = A[j][i] - d \* A[k][i];

}

B[j] = B[j] - d \* B[k];

}

}

}

for (k = nx-1; k >= 0; k--) // обратный ход

{

d = 0;

for (j = k + 1; j < nx; j++)

{

s = A[k][j] \* X[j];

d = d + s;

}

X[k] = (B[k] - d) / A[k][k];

}

return X;

}

//Решение системы классическим методом Гаусса

double\* GaussSys1(int NN, double (\*A)[N1],double B[])

{

int nx=NN, i, j, k;

double d, s;

double \*X = new double[nx];//столбец с решением

for (k = 0; k < nx; k++) // прямой ход

{

for (j = k + 1; j < nx; j++)

{

if (A[j][k]!=0)

{

d = A[j][k] / A[k][k];

for (i = k; i < nx; i++)

{

A[j][i] = A[j][i] - d \* A[k][i];

}

B[j] = B[j] - d \* B[k];

}

}

}

for (k = nx-1; k >= 0; k--) // обратный ход

{

d = 0;

for (j = k + 1; j < nx; j++)

{

s = A[k][j] \* X[j];

d = d + s;

}

X[k] = (B[k] - d) / A[k][k];

}

return X;

}

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

{

double h = (b-a)/n;

double x0;

double sum = 0;

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

{

x0=a+i\*h;

sum += f(x0);

}

sum \*= h;

return sum;

}

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

{

double h = (b-a)/n;

double x0;

double sum = 0;

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

{

x0=a+i\*h+h/2;

sum += f(x0);

}

sum \*= h;

return sum;

}

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

{

double h = (b-a)/n;

double x0, x1;

double sum = 0;

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

{

x0=a+i\*h;

x1=x0+h;

sum += f(x0) + f(x1);

}

sum \*= h/2.0;

return sum;

}

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

{

double h = (b-a)/n;

double x0, x1, x2;

double sum = 0;

for(int i=0; i<=n-2; i+=2)

{

x0=a+i\*h;

x1=x0+h;

x2=x0+2\*h;

sum += f(x0) + 4\*f(x1) + f(x2);

}

sum \*= h/3.0;

return sum;

}

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

{

double x,h = (b-a)/n;

double sum = 0;

x=a;

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

{

//определяем узлы интерполяционной формулы

double z1 = x;

double z2 = x+h;

double z15 = x+0.5\*h;

// моменты весовой фйнкции

double mu[3], C[3][3];

double \*Ak = new double[N+1];

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

{

mu[j]=(pow(b-z1,j-beta+1)- pow(b-z2,j-beta+1))/(j-beta+1);

}

//коэффициенты квадратурной формулы

C[0][0]=1;C[0][1]=1;C[0][2]=1;

C[1][0]=b-z1;C[1][1]=b-z15;C[1][2]=b-z2;

C[2][0]=(b-z1)\*(b-z1);C[2][1]=(b-z15)\*(b-z15);C[2][2]=(b-z2)\*(b-z2);

Ak = GaussSys(3, C, mu);

sum = sum + Ak[0]\*f(z1)+Ak[1]\*f(z15)+Ak[2]\*f(z2);

x=a+(i+1)\*h;

delete [] Ak;

}

return sum;

}

void NKotesE(double a, double b, double eps, int n0)

{

double s[N1], h[N1],rh[N1], C[N1][N1], J[N1-1];

int m, n=2\*n0, r = 1;

h[0]=(b-a)/n0;s[0] = NKotesN(a,b,n0);//cout<<s[0]<<endl;

h[1]=h[0]/2;s[1] = NKotesN(a,b,2\*n0);

m = 3;//порядок точности метода Ньютона-Котеса(3)

C[0][0]=1;C[0][1]=-pow(h[0],m);

C[1][0]=1;C[1][1]=-pow(h[1],m);

double \*Ak = new double[2];

Ak = GaussSys1(2, C, s);

s[1] = NKotesN(a,b,2\*n0);

J[0] = Ak[0];//уточнённый интеграл

rh[0] = J[0]-s[0];rh[1] = J[0]-s[1];//оценки погрешностей

delete [] Ak;

cout<<"Расчёт с точностью "<<eps<<" по методу Ньютона-Котеса(3): "<<endl;

while (fabs(rh[r]) > eps)

{

r = r + 1;

h[r]=h[r-1]/2; n = 2\*n; s[r] = NKotesN(a,b,n);

for (int i=0; i<=r; i++) C[i][r]=-pow(h[i],m+r-1);

C[r][0] = 1;

for (int i=1; i<r; i++) C[r][i] = -pow(h[r],m+i-1);

double \*Ak = new double[r+1];

Ak = GaussSys1(r+1, C, s);

s[r] = NKotesN(a,b,n);

J[r-1] = Ak[0];//уточнённый интеграл

rh[r] = J[r-1]-s[r];//оценка погрешности

//оценка сходимости по Эйткену

cout<<"При n = " <<n<<" порядок главного члена погрешности: "<<log((s[r]-s[r-1])/(s[r-1]-s[r-2]))/log(0.5)<<endl;

delete [] Ak;

}

cout<<"По методу Ньютона-Котеса интеграл равен "<<setprecision(10)<<s[r]<<endl;

cout<<"Число отрезков разбиения n = "<<n<<endl;

cout<<"Длина шага h = "<<h[r]<<endl;

}

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

{

double x,h = (b-a)/n;

double sum = 0;

x=a;

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

{

double z1 = x;

double z2 = x+h;

// моменты весовой фйнкции

double mu[3], C[3][3], bk[3];

double \*Ak = new double[N+1];

for (int j=0; j<2\*3; j++)

{

mu[j]=(pow(b-z1,j-beta+1)- pow(b-z2,j-beta+1))/(j-beta+1);

}

//коэффициенты узлового многочлена

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

{

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

{

C[i][j]=mu[i+j];

}

bk[i]=-mu[3+i];

}

Ak = GaussSys(3, C, bk);

//определяем узлы интерполяционной формулы по формуле Кордано

double q = 0.5\*(2\*pow(Ak[2],3)/27-Ak[2]\*Ak[1]/3+Ak[0]);

double p = (3\*Ak[1]-pow(Ak[2],2))/9;

double r = sqrt(fabs(p))\*sign(q);

double phi = acos(q/pow(r,3));

double y[3];

y[0] = -2\*r\*cos(phi/3);

y[1] = 2\*r\*cos(PI/3-phi/3);

y[2] = 2\*r\*cos(PI/3+phi/3);

for (int i=0; i<3; i++) y[i]=y[i]-Ak[2]/3;

//коэффициенты формулы Гаусса

C[0][0]=1;C[0][1]=1;C[0][2]=1;

C[1][0]=y[0];C[1][1]=y[1];C[1][2]=y[2];

C[2][0]=y[0]\*y[0];C[2][1]=y[1]\*y[1];C[2][2]=y[2]\*y[2];

Ak = GaussSys(3, C, mu);

sum = sum + Ak[0]\*f(b-y[0])+Ak[1]\*f(b-y[1])+Ak[2]\*f(b-y[2]);

x=a+(i+1)\*h;

delete [] Ak;

}

return sum;

}

void GaussE(double a, double b, double eps, int n0)

{

double s[N1], h[N1],rh[N1], C[N1][N1], J[N1-1];

int m, n=2\*n0, r = 1;

h[0]=(b-a)/n0;s[0] = GaussN(a,b,n0);//cout<<s[0]<<endl;

h[1]=h[0]/2;s[1] = GaussN(a,b,2\*n0);

m = 5;//порядок точности метода Гаусса(3)

C[0][0]=1;C[0][1]=-pow(h[0],m);

C[1][0]=1;C[1][1]=-pow(h[1],m);

double \*Ak = new double[2];

Ak = GaussSys1(2, C, s);

s[1] = GaussN(a,b,2\*n0);

J[0] = Ak[0];//уточнённый интеграл

rh[0] = J[0]-s[0];rh[1] = J[0]-s[1];//оценки погрешностей

delete [] Ak;

cout<<"Расчёт с точностью "<<eps<<" по методу Гаусса(3): "<<endl;

while (fabs(rh[r]) > eps)

{

r = r + 1;

h[r]=h[r-1]/2; n = 2\*n; s[r] = GaussN(a,b,n);

for (int i=0; i<=r; i++) C[i][r]=-pow(h[i],m+r-1);

C[r][0] = 1;

for (int i=1; i<r; i++) C[r][i] = -pow(h[r],m+i-1);

double \*Ak = new double[r+1];

Ak = GaussSys1(r+1, C, s);

s[r] = GaussN(a,b,n);

J[r-1] = Ak[0];//уточнённый интеграл

rh[r] = J[r-1]-s[r];//оценка погрешности

//оценка сходимости по Эйткену

cout<<"При n = " <<n<<" порядок главного члена погрешности: "<<log((s[r]-s[r-1])/(s[r-1]-s[r-2]))/log(0.5)<<endl;

delete [] Ak;

}

cout<<"По методу Гаусса интеграл равен "<<setprecision(10)<<s[r]<<endl;

cout<<"Число отрезков разбиения n = "<<n<<endl;

cout<<"Длина шага h = "<<h[r]<<endl;

}

int main(void)

{

setlocale(LC\_ALL, "russian");

double eps = 1e-6;

double Integ[N1][7], Er[N1][7];

cout<<"Значение интеграла"<<endl;

cout<<"n \t\t"<<"Лев пр \t\t\t"<<"Сред пр\t\t\t"<<"Трапец\t\t\t"<<"Симпсон\t\t\t"<<"Гаусс(3)\t\t"<<"Нюьют-Котес(3)"<<endl;

for (int i=1; i<N1; i++)

{

Integ[i][0]=i+1;

Integ[i][1]=rectangleL(a,b,i+1);

Integ[i][2]=rectangle(a,b,i+1);

Integ[i][3]=trapezoidal(a,b,i+1);

Integ[i][4]=simpson(a,b,i+1);

Integ[i][5]=NKotesN(a,b,i+1);

Integ[i][6]=GaussN(a,b,i+1);

for (int j=1; j<=4; j++) Er[i][j]=fabs(Integ[i][j]-Itocn);

for (int j=5; j<=6; j++) Er[i][j]=fabs(Integ[i][j]-ItocnP);

for (int j=0; j<7; j++) cout<<setprecision(10)<<Integ[i][j]<<"\t\t";

cout<<endl;

}

cout<<"Абсолютная ошибка"<<endl;

cout<<"n \t"<<"Лев пр \t\t\t"<<"Сред пр \t\t"<<"Трапец\t\t\t"<<"Симпсон\t\t\t"<<"Гаусс(3)\t\t"<<"Нюьют-Котес(3)"<<endl;

for (int i=1; i<N1; i++)

{

cout<<Integ[i][0]<<"\t";

for (int j=1; j<7; j++) cout<<setprecision(7)<<Er[i][j]<<"\t\t";

cout<<endl;

}

GaussE(a, b, eps, 1);cout<<endl;

NKotesE(a, b, eps, 1);cout<<endl;

//выбор оптимального шага разбиения для метода Ньютона-Котеса

double s1,s2,s4;

s1 = NKotesN(a,b,1);s2 = NKotesN(a,b,2);s4 = NKotesN(a,b,4);

double m = log((s4-s2)/(s2-s1))/log(0.5);

double hop = 0.25\*(b-a)\*pow(eps/fabs(ItocnP-s4),1/m);

int nopt = ceil((b-a)/hop);

double hopt = (b-a)/nopt;

cout<<"Оценка скорости сходимости: "<< m << ". n = " <<nopt<< ". Оптимальный шаг:" <<hopt<<endl;

NKotesE(a, b, eps, nopt/2);cout<<endl;

return 0;

}