|  |
| --- |
| БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ |
| Факультет прикладной математики и информатики |
| Кафедра информационных систем управления |
| Лукьянович Александр Сергеевич  10 группа, 2 курс |
| Лабораторная работа |
|  |
|  |
| **Методы численного интегрирования**    **Преподаватель**  *Полевиков Виктор Кузьмич* |
|  |
| Дата сдачи:  23.05.2017 |

Оглавление

[Постановка задачи 3](#_Toc483371154)

[Краткая теория 4](#_Toc483371155)

[1) 4](#_Toc483371156)

[2) 4](#_Toc483371157)

[3) 5](#_Toc483371158)

[4) 5](#_Toc483371159)

[Листинг программы 7](#_Toc483371160)

[Результаты 9](#_Toc483371161)

# Постановка задачи

Интеграл имеет вид

Вычислить интеграл с точностью , число разбиений N определив по правилу Рунге, методом:

1. Левых прямоугольников
2. Трапеций
3. Симпсона
4. НАСТ (АСТ=0, 1, 2)

# Краткая теория

Пусть есть квадратурная формула порядка точности и она даёт точное значение , а приближенное значение этого интеграла, полученное с шагом , обозначим

Легко видеть, что . Если , то . Если

То точность достигнута и можно принять , иначе и продолжаем.

В нашем случае вес , поэтому формулы примут упрощённый вид

## 1)

Отрезок дробится с достаточной точностью, а сам интеграл вычисляется

Но подынтегральная функция имеет неопределённость в точке , поэтому избавимся от неопределённости

## 2)

В случае равномерной сетки известна формула

От неопределённости избавляемся аналогично прошлому пункту.

## 3)

Возьмём самый простой способ: число N кратное 2:

## 4)

АСТ = 2n + 1

Чтобы обойтись без длительных подсчётов и , мы изменим пределы интегрирования на [-1;1] с помощью замены:

Тогда наш интеграл примет вид

В этом случае корни многочлена Лежандра и коэффициенты посчитаны:

n = 0:

n = 1:

n = 2:

Подынтегральное выражение имеет неопределённость в точке . Избавимся от неё:

# Листинг программы

Программа выполнена на языке программирования C++, в среде Microsoft Visual Studio Community 2015.

#include "resources.h"

int main(int argc, char argv[]) {

setlocale(LC\_ALL, ".1251");

ofstream fout("output.txt", ios\_base::trunc);

if (!fout.good())

throw exception("");

fout << fixed << setprecision(13);

const double eps = pow(10, -5);

double(\*method)(int) = nullptr;

method = leftRectagle;

fout << "левых прямоугольников:\n" << runge(eps, method, 2) << endl;

method = trapeze;

fout << "\nтрапеций:\n" << runge(eps, method, 2) << endl;

method = simpson;

fout << "\nсимпсона:\n" << runge(eps, method, 4) << endl;

fout << "\nНАСТ n=0:\n" << 2 \* f\_t(0) << endl;

fout << "\nНАСТ n=1:\n" << f\_t(-1.0 / sqrt(3)) + f\_t(1.0 / sqrt(3)) << endl;

fout << "\nНАСТ n=2:\n" << 5.0 / 9.0\*f\_t(-sqrt(3.0 / 5.0)) + 8.0 / 9.0\*f\_t(0) +

5.0 / 9.0\*f\_t(sqrt(3.0 / 5.0)) << endl;

fout.close();

return 0;

}

double runge(const double eps, double(\*method)(int), int m) {

int N = 1;

double I, I\_next, alpha;

do {

I = method(N);

I\_next = method(2 \* N);

alpha = (I\_next - I) / (pow(h(N), m) - pow(h(2 \* N), m));

N \*= 2;

} while (fabs(I\_next - I) / (pow(2, m) - 1) > eps);

return I\_next + alpha\*pow(h(N), m);

}

double leftRectagle(int N) {

double sum = 0.0;

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

sum += f(x(i, N));

return sum \* h(N);

}

double trapeze(int N) {

double sum = 0.5\*(f(x(0, N)) + f(x(N, N)));

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

sum += f(x(i, N));

return sum\*h(N);

}

double simpson(int N) {

double sum = 0.0;

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

sum += f(x(i - 1, N)) + 4 \* f(x(i, N)) + f(x(i + 1, N));

return sum\*h(N) / 3;

}

double f(double x) {

if (x == 0.0)

return 1.0;

else

return log(1 + x) / x;

}

double f\_t(double t) {

if (t == -1)

return 0.5;

else

return log(0.5\*(3 + t)) / (t + 1);

}

double x(int i, int N) { return h(N)\*i; }

double h(int N) { return 1.0 / N; }

# Результаты

левых прямоугольников:

N = 16384

приближённое значение: 0.8224670332336

погрешность:

трапеций:

N = 64

приближённое значение: 0.8224670338680

погрешность:

симпсона:

N = 8

приближённое значение: 0.8224671596293

погрешность:

НАСТ n = 0:

приближённое значение: 0.8109302162163

погрешность: -0.0115368172078

НАСТ n = 1:

приближённое значение: 0.8222423320350

погрешность: -0.0002247013891

НАСТ n = 2:

приближённое значение: 0.8224620966512

погрешность: -0.0000049367729