МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Национальный исследовательский
Нижегородский государственный университет им. Н.И. Лобачевского»
(ННГУ)
Институт информационных технологий, математики и механики
Направление подготовки: «Фундаментальная информатика и информационные технологии»

ОТЧЕТ

# Методы численного интегрирования

Выполнил: студент группы 382006-2

Сухарев Артём Андреевич _____
подпись
Эгамов Альберт Исмаилович _____
подпись

Нижний Новгород

2022 г.

# Содержание

## Введение

Задача численного интегрирования заключается в нахождении значения определенного интеграла, с помощью некоторых методов.

Необходимость применять подобное вычисление интеграла возникает в нескольких случаях: когда подынтегральная функция не задана аналитически, когда для данной функции невозможно найти первообразную для вычисления значения по формуле Ньютона-Лейбница или даже когда вычисление этой первообразной очень сложно и трудозатратно.

Сами же сферы применения настолько разнообразны (от применения в исследованиях и разного рода вычислений и симуляций в физике и химии, до некоторых аспектов экономики и анализа данных), что потребность в данных методах невозможно переоценить.

## Постановка задачи

При выполнении данной лабораторной работы я хочу изучить три метода численного интегрирования: формула Симпсона(метод парабол), формула «трех восьмых»(частный случай метода Ньютона-Котеса) и четырехинтервальную формулу Боде - а так же какую точность могут обеспечить данные методы при вычислении ими реальных функций для разных разбиений отрезка интегрирования.

Для этого мне необходимо реализовать программу на языке программирования python, в которой для разного количества отрезков в разбиении я смогу оценить среднюю относительную погрешность на некотором наборе функций.

## Способ задания функции

Во-первых, когда мы определились, что мы будем рассматривать набор случайных функций, нам необходим способ их задания. В силу необязательности рассмотрения всех возможных функций, можно взять какой-нибудь шаблон функции с некоторым количеством параметров. Тем не менее для большей объективности, по возможности нужно скомбинировать сразу несколько. Для этих целей я возьму следующий шаблон:

$$f(x) = \mathrm{a} * \sin(\mathrm{b} * \mathrm{x}) + \mathrm{c} * \cos(\mathrm{d} * \mathrm{x}) + \mathrm{e} + \mathrm{f} * x + g * x^2 + h * x^3 + k * x^4$$

В данном случае параметрами будут a,b,c,d,e,f,g,h,k (9 штук) и при разных значениях мы будем получать различные функции, и на разных участках x доминировать(задавать поведение) будут разные из них. В коде это можно реализовать следующим образом:

```
def gen_func():
    arr=[]
    for i in range(9):
        arr.append(random.uniform(-10,10))
    return arr

def function(arr,x):
    res = 0.0
    res += arr[0]*sin(arr[1]*x)
    res += arr[2]*cos(arr[3]*x)
    res += arr[4]+arr[5]*x+arr[6]*(x**2)+arr[7]*(x**3)+arr[8]*(x**4)
    return res

def antiderivative_function(arr,x):
    res = 0.0
    res += -arr[0]*cos(arr[1]*x)/arr[1]
    res += arr[2]*sin(arr[3]*x)/arr[3]
    res += arr[4]*x+arr[5]*(x**2)/2+arr[6]*(x**3)/3+arr[7]*(x**4)/4+arr[8]*(x**5)/5
    return res
```

То есть функция будет задаваться набором из 9 чисел, и по данному набору мы легко можем вычислить значение функции и её первообразную в какой-то точке.

Во-вторых, в большинстве случаев нам дается не сама функция, а лишь набор ее точек (абсцисс и ординат), и для большей реалистичности численный интеграл нужно вычислять именно по сформированному набору точек.

Для создания этого набора, нам нужно определиться с количеством интервалов, которые мы будем рассматривать и границы самого отрезка. Не уменьшая общности можно рассматривать отрезок [-40,40]. Но проблема может возникнуть с количеством интервалов, не вдаваясь пока в подробности

(это будет представлено далее) можно сказать, что одно вычисление для каждого метода учитывает разное число интервалов (2, 3 или 4), поэтому нам нужно брать только такое количество интервалов, которое кратно всем. Несложно заметить, что для этого нужно брать только кратные 12 количества (12, 24, 36 и т.д.).

Когда же мы определились со всеми нужными значениями, можно преступить к вычислению, а конкретно если границы отрезка [l,r], а число интервалов n, то точки будут иметь следующие значения абсцисс:

$$X = l + i * \frac{r-l}{n}, i = \overline{1, n+1}$$

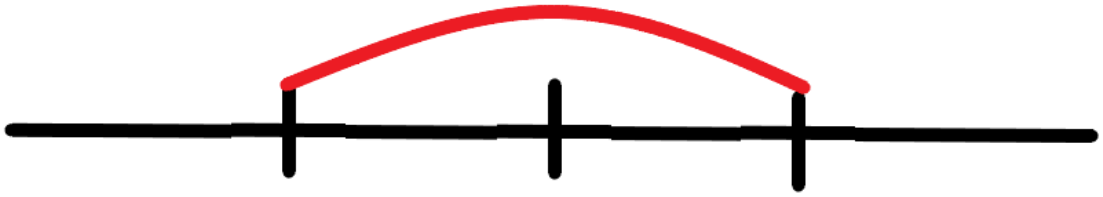
```
def gen_vector(arr,l,r,n):
    vector=[]
    len = (r-l)/n
    for i in range(n+1):
        vector.append(function(arr,l+i*len))
    return vector
```

## Формулы и методы численного интегрирования

**Формула Симпсона**

Этот метод подсчета является двухинтервальным, то есть для нахождения значения нужно разбить отрезок на два интервала тремя точками. Поскольку нам может быть дано большее число интервалов и точек, то их можно разбить на блоки по нужному нам количеству и считать на каждом блоке по 2 интервала отдельно, а позже просуммировать полученные значения.

Теперь рассмотрим один блок, состоящий из двух интервалов:



Пусть у нас есть 3 точки $x_i, x_{i+1}$ и $x_{i+2}$ и посчитанные в них значения $f_i, f_{i+1}$ и $f_{i+2}$. Тогда интеграл в этом блоке можно посчитать следующим образом:

$$I_i = \int_{x_i}^{x_{i+2}} f(x)dx = \frac{h}{3}(f_i + 4 * f_{i+1} + f_{i+2}), \text{где } h = \frac{r-l}{n} \text{длина интервала}$$

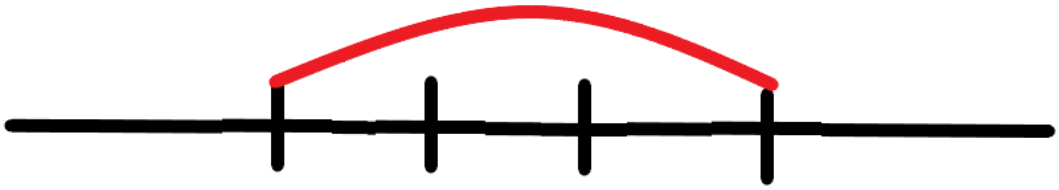
Сумма по всем $\frac{n}{2}$ блокам и даст нам интеграл, который мы хотим найти:

$I = \sum_{i=0}^{\frac{n}{2}-1} I_{i*2}$

```
def simpson(arr,n,len):
    res = 0.0
    for i in range(0,(n+1)-2,2):
        res += arr[i] + 4*arr[i+1] + arr[i+2]
    res = res * len * 2 / 6
    return res
```

**Формула «трех восьмых»**

Это трехинтервальная формула на четырехточечном шаблоне. В данном случае блок будет рассматриваться из трех интервалов.



Итак теперь у нас есть 4 точки $x_i, x_{i+1}, x_{i+2}$ и $x_{i+3}$ и посчитанные в них значения $f_i, f_{i+1}, f_{i+2}$ и $f_{i+3}$. Тогда интеграл в этом блоке можно посчитать следующим образом:

$$I_i = \int_{x_i}^{x_{i+3}} f(x)dx = \frac{3h}{8}(f_i + 3 * f_{i+1} + 3 * f_{i+2} + f_{i+3}),$$

$$\text{где } h = \frac{r-l}{n} \text{длина интервала}$$

А также поскольку количество блоков изменится, то и сумма тоже:

$$I = \sum_{i=0}^{\frac{n}{3}-1} I_{i*3}$$

```
def three_eights(arr,n,len):
    res = 0.
    for i in range(0,(n+1)-3,3):
        res += arr[i] + 3*arr[i+1] + 3*arr[i+2] + arr[i+3]
    res = res * len * 3 / 8
    return res
```

**Четырехинтервальная формула Боде**

Это четырехинтервальная формула на пятиточечном шаблоне. То есть в данном случае блок будет рассматриваться из четырех интервалов.

В данной формуле у нас есть 5 точек $x_i, x_{i+1}, x_{i+2}, x_{i+3}$ и $x_{i+4}$ и посчитанные в них значения $f_i, f_{i+1}, f_{i+2}, f_{i+3}$ и $f_{i+4}$. Тогда интеграл в этом блоке можно посчитать следующим образом:

$$I_i = \int_{x_i}^{x_{i+4}} f(x)dx = \frac{4h}{90}(7 * f_i + 32 * f_{i+1} + 12 * f_{i+2} + 32 * f_{i+3} + 7 * f_{i+4}),$$

где $h = \frac{r-l}{n}$ длина интервала

Аналогично изменится и сумма:

$$I = \sum_{i=0}^{\frac{n}{4}-1} I_{i*4}$$

```
def five_points(arr,n,len):
    res = 0.0
    for i in range(0,(n+1)-4,4):
        res += 7*arr[i] + 32*arr[i+1] + 12*arr[i+2] + 32*arr[i+3] + 7*arr[i+4]
    res = res * len * 4 / 90
    return res
```

## Производство вычислений

Для проведения сверок с верным значением определенного интеграла нам необходимо его вычислить, а так как у нас определена функция вычисления первообразной в точке, то можно вычислить интеграл с помощью формулы Ньютона-Лейбница.

```
def right_integr(arr,l,r):
    return antiderivative_function(arr,r)-antiderivative_function(arr,l)
```

Теперь можем создать функцию, которая проводит серию тестов и вычисляет среднюю относительную погрешность. Создадим выборку из 5000 тестовых случайных функций и посчитаем эту погрешность:

$$R = \frac{\sum_{i=0}^{5000} \left| \frac{best-curr}{best} \right|}{5000}$$

```
def calc_for_num_lines(n,flag):
    k=5000
    a,b,c = 0,0,0
    r = 40
    l = -40
    for i in range(k):
        arr = gen_func()
        vector = gen_vector(arr, l, r, n)
        best = right_integr(arr, l, r)

        simp = simpson(vector, n, (r-l)/n)
        te = three_eights(vector, n, (r-l)/n)
        fp = five_points(vector, n, (r-l)/n)

        a+=abs((simp-best)/best)
        b+=abs((te-best)/best)
        c+=abs((fp-best)/best)
    a/=k
    b/=k
    c/=k
    if flag==1:
        return a
    if flag==2:
        return b
    if flag==3:
        return c
```

На вход функции так же подадим специальный флаг, который будет показывать какой именно из результатов нам нужен.

## Представление результатов и используемые технические средства

Для визуализации результатов я загнал полученные данные в виде графика и таблицы.

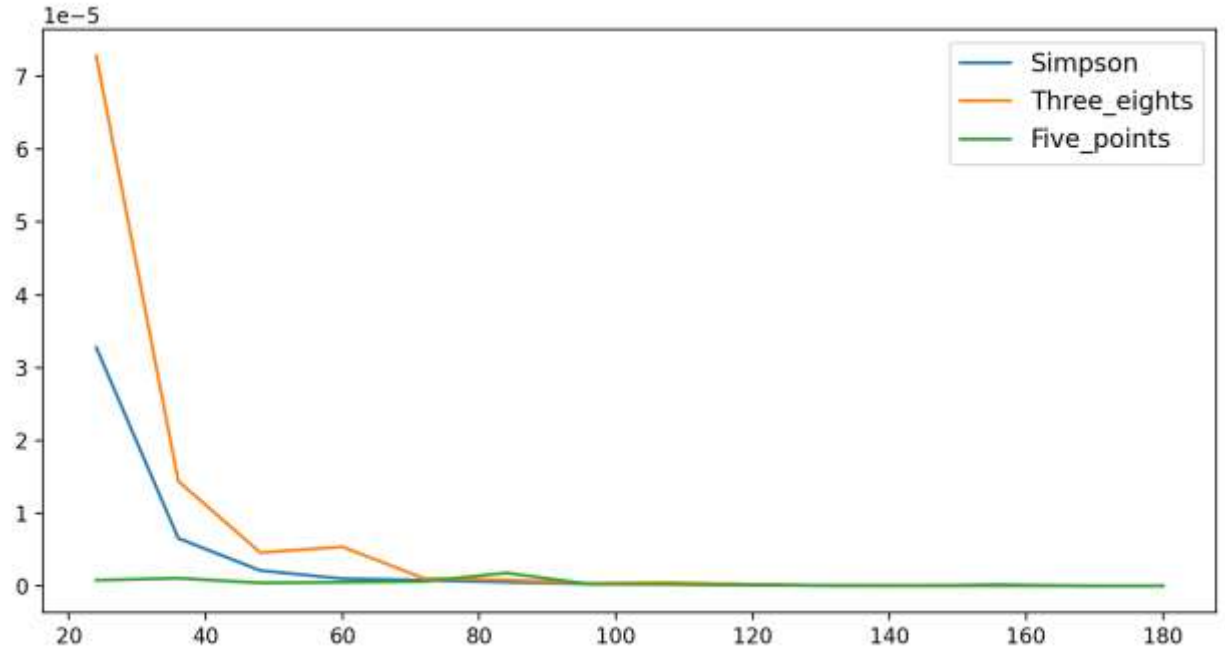
```
x = arange(24, 181, 12)
simpsonarr = [calc_for_num_lines(i,1) for i in x]
tearr = [calc_for_num_lines(i,2) for i in x]
fparr = [calc_for_num_lines(i,3) for i in x]

plt.figure(figsize=(10, 5))
plt.plot(x, simpsonarr, label='Simpson')
plt.plot(x, tearr, label='Three_eights')
plt.plot(x, fparr, label='Five_points')
plt.legend(loc='best', fontsize=12)
plt.show()

table = PrettyTable(["Кол-во интервалов","Формула Симпсона","Формула \"трех восьмых\"","Четырехинтервальная формула Боде"])

for i in range(len(simpsonarr)):
    table.add_row([x[i],simpsonarr[i],tearr[i],fparr[i]])
print(table)
```

При запуске программы будет показан график с полученными значениями средней относительной погрешности для каждого метода.




Так же данные будут выведены в таблицу, так как график не может в полном масштабе отобразить погрешности при больших количествах интервалов.

| Кол-во интервалов | Формула Симпсона | Формула "трех восьмых" | Четырехинтервальная формула Боде |
|---|---|---|---|
| 24 | 3.27688175245744e-05 | 7.27945369342977e-05 | 8.391216491109294e-07 |
| 36 | 6.585011877407569e-06 | 1.4418558939355947e-05 | 1.1099125820275123e-06 |
| 48 | 2.1696087329836253e-06 | 4.62785253407116e-06 | 4.708564557610132e-07 |
| 60 | 1.04969600065069e-06 | 5.4210416911836384e-06 | 5.516201475522363e-07 |
| 72 | 8.466943886830738e-07 | 1.0694338001266955e-06 | 6.755035750241294e-07 |
| 84 | 5.662475808408027e-07 | 8.774902722455966e-07 | 1.8402912193493145e-06 |
| 96 | 3.387708242885494e-07 | 4.252611658969434e-07 | 3.0997665602997117e-07 |
| 108 | 4.480305825617925e-07 | 5.293082567059864e-07 | 2.6218665723184286e-07 |
| 120 | 2.500952573386287e-07 | 2.3502488191328841e-07 | 1.678558795928935e-07 |
| 132 | 9.821448090916796e-08 | 1.231577002269926e-07 | 1.1191774626196133e-07 |
| 144 | 8.7804822045704e-08 | 9.398854937172510-08 | 7.940907947197127e-08 |
| 156 | 2.1677096264138565e-07 | 7.679487620077179e-08 | 1.1980167161455374e-07 |
| 168 | 5.0783227606130764e-08 | 7.589960449397044e-08 | 5.2395052173942296e-08 |
| 180 | 5.164517833525874e-08 | 5.574347270254278e-08 | 6.091204933850562e-08 |

Так же стоит заметить, что для получения случайных значений, создания таблицы и графика я использовал дополнительные модули. А именно модули numpy, math, matplotlib и prettytable, поэтому при необходимости запуска стоит убедится, что они у вас установлены.

Так же стоит учесть, что набор функций для тестов большой и для выполнения программы нужно определенное время. На моем компьютере это занимает около минуты(если вам нужна быстрая программа снизьте количество тестов, то есть значение k).

## Результаты работы

По проведенным результатам можно увидеть, что даже при небольших количествах интервалов и точек можно достичь хорошей оценки интеграла.

Так же по данным таблицы и графика можно увидеть, что формула Боде дает меньшую погрешность по сравнению с формулой Симпсона, а формула Симпсона лучшую погрешность по сравнению с методом «трех восьмых». Причем данная тенденция распространяется при всех количествах интервалов.

Отсюда можно сделать вывод, что самый вариант из этих трех это четырехинтервальная формула Боде.

## Литература


1) Лекция. Методы численного интегрирования. http://dep805.ru/education/portal/4/to/l13_4tochm.pdf
2) https://cyclowiki.org/wiki/Формула_трёх_восьмых
3) https://ru.wikipedia.org/wiki/Численное_интегрирование

# Исходный код

```
from numpy import random,arange

from math import sin,cos

import matplotlib.pyplot as plt

from prettytable import PrettyTable


# function = arr[0] * sin(arr[1]*x) + arr[2] * cos(arr[3]*x) +

#            + arr[4] + arr[5]*x + arr[6]*x^2 + arr[7]*x^3 + arr[8]*x^4


def gen_func():

   arr=[]

   for i in range(9):

      arr.append(random.uniform(-10,10))

   return arr


def function(arr,x):

   res = 0.0

   res += arr[0]*sin(arr[1]*x)

   res += arr[2]*cos(arr[3]*x)

   res += arr[4]+arr[5]*x+arr[6]*(x**2)+arr[7]*(x**3)+arr[8]*(x**4)

   return res


def antiderivative_function(arr,x):

   res = 0.0

   res += -arr[0]*cos(arr[1]*x)/arr[1]

   res += arr[2]*sin(arr[3]*x)/arr[3]

   res += arr[4]*x+arr[5]*(x**2)/2+arr[6]*(x**3)/3+arr[7]*(x**4)/4+arr[8]*(x**5)/5

   return res
```

```
def gen_vector(arr,l,r,n):
    vector=[]
    len = (r-l)/n
    for i in range(n+1):
        vector.append(function(arr,l+i*len))
    return vector


def right_integr(arr,l,r):
    return antiderivative_function(arr,r)-antiderivative_function(arr,l)


def simpson(arr,n,len):
    res = 0.0
    for i in range(0,(n+1)-2,2):
        res += arr[i] + 4*arr[i+1] + arr[i+2]
    res = res * len * 2 / 6
    return res


def three_eights(arr,n,len):
    res = 0.
    for i in range(0,(n+1)-3,3):
        res += arr[i] + 3*arr[i+1] + 3*arr[i+2] + arr[i+3]
    res = res * len * 3 / 8
    return res


def five_points(arr,n,len):
    res = 0.0
    for i in range(0,(n+1)-4,4):
        res += 7*arr[i] + 32*arr[i+1] + 12*arr[i+2] + 32*arr[i+3] + 7*arr[i+4]
```

```
    res = res * len * 4 / 90
    return res


def calc_for_num_lines(n,flag):
    k=5000
    a,b,c = 0,0,0
    r = 40
    l = -40
    for i in range(k):
        arr = gen_func()
        vector = gen_vector(arr, l, r, n)
        best = right_integr(arr, l, r)


        simp = simpson(vector, n, (r-l)/n)
        te = three_eights(vector, n, (r-l)/n)
        fp = five_points(vector, n, (r-l)/n)


        a+=abs((simp-best)/best)
        b+=abs((te-best)/best)
        c+=abs((fp-best)/best)
    a/=k
    b/=k
    c/=k
    if flag==1:
        return a
    if flag==2:
        return b
    if flag==3:
```

```
    return c


x = arange(24, 181, 12)
simpsonarr = [calc_for_num_lines(i,1) for i in x]
tearr = [calc_for_num_lines(i,2) for i in x]
fparr = [calc_for_num_lines(i,3) for i in x]

plt.figure(figsize=(10, 5))
plt.plot(x, simpsonarr, label='Simpson')
plt.plot(x, tearr, label='Three_eights')
plt.plot(x, fparr, label='Five_points')
plt.legend(loc='best', fontsize=12)
plt.show()

table = PrettyTable(["Кол-во интервалов","Формула Симпсона","Формула \"трех
восьмых\"","Четырехинтервальная формула Боде"])

for i in range(len(simpsonarr)):
   table.add_row([x[i],simpsonarr[i],tearr[i],fparr[i]])
print(table)
```