**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет прикладной математики и физики

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Численные методы»

**Курсовой проект**

|  |  |  |
| --- | --- | --- |
|  | Студент: | Чекушкин Д.И. |
| Группа: | | 8О-304Б |
| Преподаватель: | | Иванов И.Э. |
|  | |  |
| Оценка: | |  |
| Дата: | |  |

Москва 2019

# Постановка задачи

Вычисление многократных интегралов с использованием квадратурных формул и метода Монте-Карло.

# Описание методов:

### **Алгоритм интегрирования по ММК**

Идея ММК для численного интегрирования  заключается в использовании теоремы о среднем из математического анализа, в которой утверждается, что интеграл    равен произведению длины отрезка (здесь b−a) и среднего значения f¯ функции f на отрезке [a,b]. Среднее значение может быть вычислено с помощью выборки значений f на множестве случайных точек внутри области и вычислении их арифметического среднего. В многомерном случае, интеграл оценивается как произведение площади (объема) области и среднего значения функции, которое опять вычисляется по выборке на множестве случайных точек.

Введем некоторые величины, которые позволят нам формализовать алгоритм численного интегрирования. Пусть дан двумерный интеграл:

где Ω — двумерная область заданная посредством вспомогательной функции g(x,y):

Ω={(x,y) : g(x,y)≥0}.

Таким образом, граница области ∂Ω задана неявной функцией (кривой) g(x,y)=0.

Такое описание областей распространено в последние десятилетия, при этом g называется функцией уровня, а граница g=0 — нулевым контуром функции уровня.

Для простых областей можно легко построить функцию g вручную, но в более сложных промышленных приложениях следует обратиться к математическим моделям построения g.

Пусть A(Ω)— площадь области Ω. Мы можем численно найти интеграл по следующему ММК:

1. помещаем область Ω внутрь прямоугольника R;
2. генерируем большое число случайных точек на R;
3. вычисляем долю q точек, которые попали в область Ω;
4. приближаем A(Ω)/A(R) числом q, т.е., полагаем A(Ω)=qA(R);
5. вычисляем среднее значение f¯ функции f на области Ω;
6. вычисляем приближенное значение интеграла как A(Ω)f¯.

Отметим, что площадь A(R) прямоугольника R легко вычислить, при том что площадь A(Ω) нам не известна. Однако, если предположить, что доля площади A(R) занимаемой областью Ω такая же как доля случайных точек, попавших внутрь Ω , можно получить простое приближение для A(Ω).

## **Метод парабол .**

Пусть функция y = f(x) непрерывна на отрезке [a; b] и нам требуется вычислить определенный интеграл .

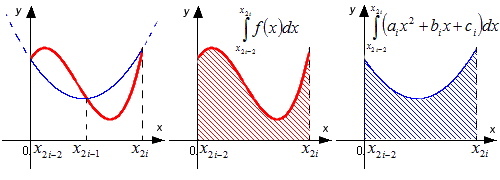
Разобьем отрезок [a; b] на n элементарных отрезков   длины точками . Пусть точки формулаявляются серединами отрезков формула соответственно. В этом случае все "узлы" определяются из равенства формула.

### Суть метода парабол.

На каждом интервале формула подынтегральная функция приближается квадратичной параболой , проходящей через точки формула. Отсюда и название метода - метод парабол.

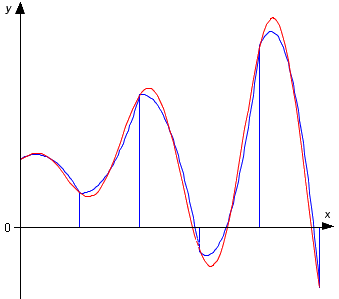
Это делается для того, чтобы в качестве приближенного значения определенного интеграла  . Взять формула, который мы можем вычислить по формуле Ньютона-Лейбница. В этом и заключается **суть метода парабол**.

Геометрически это выглядит так:

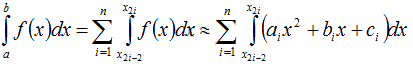


### Графическая иллюстрация метода парабол (Симпсона).

Красной линией изображен график функции y=f(x), синей линией показано приближение графика функции y=f(x) квадратичными параболами на каждом элементарном отрезке разбиения.



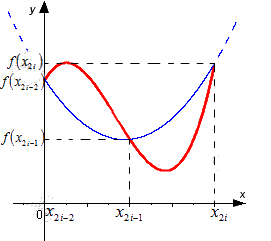
### Вывод формулы метода Симпсона (парабол).

В силу [свойств определенного интеграла](http://www.cleverstudents.ru/integral/definite_integral_properties.html) имеем .

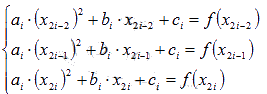
Для получения формулы метода парабол (Симпсона) нам осталось вычислить формула.

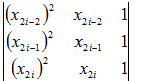
Пусть формула (мы всегда можем к этому прийти, проведя соответствующее геометрическое преобразования сдвига для любого i = 1, 2, ..., n).

Сделаем чертеж.

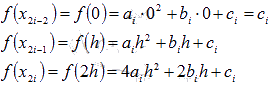


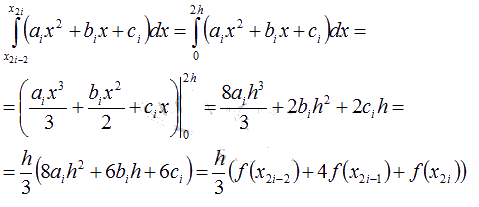
Покажем, что через точки  формулапроходит только одна квадратичная парабола . Другими словами, докажем, что коэффициенты формула определяются единственным образом.

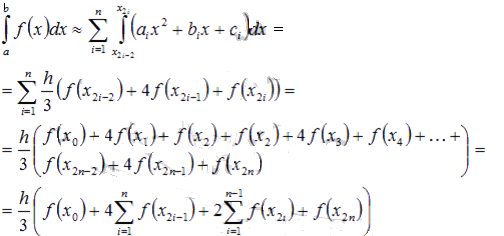
Так как формула - точки параболы, то справедливо каждое из уравнений системы  


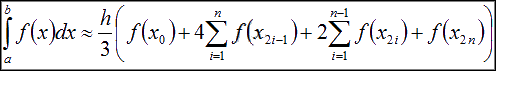
Записанная система уравнений есть система линейных алгебраических уравнений относительно неизвестных переменных формула. Определителем основной матрицы этой системы уравнений является определитель Вандермонда , а он отличен от нуля для несовпадающих точек формула. Это указывает на то, что система уравнений имеет единственное решение (об этом говорится в статье [решение систем линейных алгебраических уравнений](http://www.cleverstudents.ru/systems/solving_systems_of_linear_equations.html)), то есть, коэффициенты формула определяются единственным образом, и через точки формула проходит единственная квадратичная парабола.

Перейдем к нахождению интеграла формула.

Очевидно:  


Используем эти равенства, чтобы осуществить последний переход в следующей цепочке равенств:  


Таким образом, можно получить формулу метода парабол:  


**Формула метода Симпсона (парабол)** имеет вид  
.

# Описание работы программы

После запуска, программа по алгоритму проводит численное интегрирование заранее заданных функций. На вывод подается результат в виде таблицы.

Программа вычисляет два интеграла.

Первый:

h = 0.1 num= 10

Кратность Метод Монте-Карло Метод Квадратур Аналитически

1 0.531381083161214 0.500012469860719 0.5

2 65.56911143639249 55.96844163586613 58.18253870833333

3 1807.9652885179373 1750.484289852044 1819.73299572328

4 1074.3405853610645 1214.0866816515802 1262.1149348320193

5 455150.3324981824 2375057.0709809037 2469012.341265138

h = 0.01 num= 100

Кратность Метод Монте-Карло Метод Квадратур Аналитически

1 0.5114515156831498 0.5000000012911946 0.5

2 56.29449304047427 57.93143464960154 58.18253870833333

3 1623.0346517678574 1811.8793964960955 1819.73299572328

4 1181.016987755989 1256.6679023426018 1262.1149348320193

5 2862098.5760990544 2458356.583957715 2469012.341265138

h = 0.0001 num= 10000

Кратность Метод Монте-Карло Метод Квадратур Аналитически

1 0.501411642313671 0.5000000000000001 0.5

2 58.530573446130184 58.18253870833341 58.18253870833333

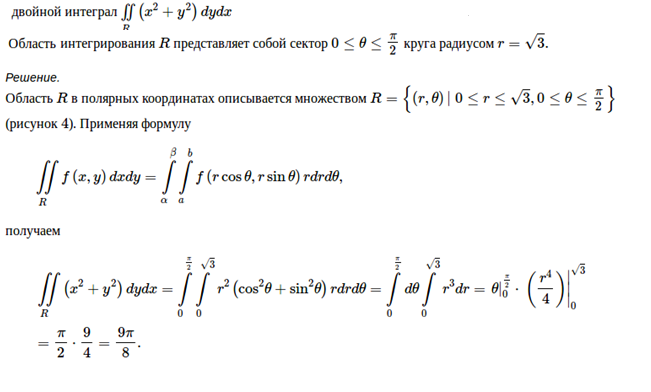
3 1817.2286342196255 1819.7329957232837 1819.73299572328

4 1263.9540747859871 1262.1149348320207 1262.1149348320193

5 2465965.4064612756 2469012.3412651415 2469012.341265138

Можно заметить, что при увеличении количества экспериментов и уменьшением шага h, результаты вычисления становятся точнее

Второй:



Результаты получились довольно близкие:

С помощью метода Монте-Карло 3.519646818291589

С помощью квадратур 3.620334371559588

Вычисленный аналитически 3.5342917352885173

Третий:

Где Ω круг с центром в начале координат и радиусом 2, и пусть f(x,y)=

Точное значение интеграла: 16.7551608191456

Вычисленное: 16.656490520940682

Код программы

import numpy as np

import math

import matplotlib as mpl

import matplotlib.pyplot as plt

def Monte\_carlo\_method(intervals, f, experiments\_amount):

size = len(intervals)

left, right = zip(\*intervals)

vol=[]

for element in intervals:

vol.append(element[1]-element[0])

#print('vol',vol)

volume=vol[0]

#print(volume)

#print(vol)

for s in vol[1:]:

#print(s)

volume=volume\*s

integral = volume \* np.sum([f(np.random.uniform(left, right, size)) for \_ in range(experiments\_amount)]) / experiments\_amount

return integral

def Quadrature\_method(interval, f, h):

#Метод парабол

a, b = interval

n = int((b - a) / h + 1)

integral = h \* (f(a) + 4 \* np.sum([f(a + i \* h) for i in range(1, n, 2)])+ 2 \* np.sum([f(a + i \* h) for i in range(2, n - 1, 2)]) + f(b))/3

return integral

def Func\_interval(f, interval):

a, b = interval

return f(b) - f(a)

def f1(x):

return (1/(x\*\*2))

def f2(x):

return (x\*\*2)

def f3(x):

return (math.sqrt(x))

def f4(x):

return (math.sin(x))

def f5(x):

return (x\*\*3)

funcs = [f1,f2,f3,f4,f5]

def af1(x):

return (-1/(x))

def af2(x):

return ((x\*\*3)/3)

def af3(x):

return ((2\*x\*\*(1.5))/3)

def af4(x):

return (-math.cos(x))

def af5(x):

return ((x\*\*4)/4)

analitical\_functions=[af1,af2,af3,af4,af5]

monte\_funcs = [f1, lambda x: f1(x[0]) \* f2(x[1]), lambda x: f1(x[0]) \* f2(x[1]) \* f3(x[2]), lambda x: f1(x[0]) \* f2(x[1]) \* f3(x[2]) \* f4(x[3]), lambda x: f1(x[0]) \* f2(x[1]) \* f3(x[2]) \* f4(x[3]) \* f5(x[4])]

intervals = [(1, 2), (2.005, 7.095), (5, 15),(8, 10),(12, 13)]

h = 0.1

numbers = [10,100,10000]

all\_errorsM=[]

all\_errorsK=[]

for num in numbers:

quad\_result = []

monte\_result = []

analitical\_result = []

errorM=[]

errorK=[]

for i in range(len(intervals)):

j = i + 1

#print(intervals[:j])

quad\_ints = np.prod([Quadrature\_method(interval, func, h) for func, interval in zip(funcs[:j], intervals[:j])])

quad\_result.append(quad\_ints)

monte\_int = Monte\_carlo\_method(intervals[:j], monte\_funcs[i], num)

monte\_result.append(monte\_int)

analitical\_result.append(np.prod([Func\_interval(func, interval) for func, interval in zip(analitical\_functions[:j], intervals[:j])]))

errorM.append(analitical\_result[i]-monte\_result[i])

errorK.append(analitical\_result[i]-quad\_result[i])

all\_errorsM.append(errorM)

all\_errorsK.append(errorK)

#print("ERROR\_M=",errorM)

#print("ERROR\_K=",errorK)

print('h = ',h,'num=',num)

print("Кратность ", "Метод Монте-Карло", " Метод Квадратур", " Аналитически")

for i in range(5):

print(' ',i+1,' ', monte\_result[i],' ', quad\_result[i],' ', analitical\_result[i])

h=h\*h

print("all\_errorsM",all\_errorsM)

print("all\_errorsK",all\_errorsK)

#############################

# Второй пример

intervals = [(0, math.pi/2), (0, math.sqrt(3))]

def f1(x):

return int(x\*\*0)

def f2(x):

return int(x\*\*3)

funcs = [f1,f2]

monte\_funcs = [f1, lambda x: f1(x[0]) \* f2(x[1])]

circle\_int = Monte\_carlo\_method(intervals[:2], monte\_funcs[1], 100000)

print('С помощью метода Монте-Карло', circle\_int)

int1=Quadrature\_method(intervals[1], funcs[1], 0.1)

int2=Quadrature\_method(intervals[0], funcs[0], 0.1)

print('С помощью квадратур', int2\*int1)

# результат, полученный аналитически

print('Вычесленный аналитически', (9\*math.pi)/8)

x\_vals=[1,2]

y\_first=all\_errorsM[0][0:2]

y\_second=all\_errorsM[1][0:2]

y\_third=all\_errorsM[2][0:2]

plt.plot(x\_vals, y\_first, color='r')

plt.plot(x\_vals, y\_second, color='y')

plt.plot(x\_vals, y\_third, color='g')

plt.legend ( ("n = 10", "n = 100","n = 10000") )

plt.show()

plt.savefig('graph1.png')

x\_vals=[1,2,3,4]

y\_first=all\_errorsM[0][0:4]

y\_second=all\_errorsM[1][0:4]

y\_third=all\_errorsM[2][0:4]

plt.plot(x\_vals, y\_first, color='r')

plt.plot(x\_vals, y\_second, color='y')

plt.plot(x\_vals, y\_third, color='g')

plt.legend ( ("n = 10", "n = 100","n = 10000") )

plt.show()

plt.savefig('graph2.png')

x\_vals=[1,2,3,4,5]

y\_first=all\_errorsM[0][0:5]

y\_second=all\_errorsM[1][0:5]

y\_third=all\_errorsM[2][0:5]

plt.plot(x\_vals, y\_first, color='r')

plt.plot(x\_vals, y\_second, color='y')

plt.plot(x\_vals, y\_third, color='g')

plt.legend ( ("n = 10", "n = 100","n = 10000") )

plt.show()

plt.savefig('graph3.png')

x\_vals=[1,2]

y\_first=all\_errorsK[0][0:2]

y\_second=all\_errorsK[1][0:2]

y\_third=all\_errorsK[2][0:2]

plt.plot(x\_vals, y\_first, color='r')

plt.plot(x\_vals, y\_second, color='y')

plt.plot(x\_vals, y\_third, color='g')

plt.legend ( ("h = 0.1", "h = 0.01","h = 0.0001") )

plt.show()

plt.savefig('graph4.png')

x\_vals=[1,2,3,4]

y\_first=all\_errorsK[0][0:4]

y\_second=all\_errorsK[1][0:4]

y\_third=all\_errorsK[2][0:4]

plt.plot(x\_vals, y\_first, color='r')

plt.plot(x\_vals, y\_second, color='y')

plt.plot(x\_vals, y\_third, color='g')

plt.legend ( ("h = 0.1", "h = 0.01","h = 0.0001") )

plt.show()

plt.savefig('graph5.png')

x\_vals=[1,2,3,4,5]

y\_first=all\_errorsK[0][0:5]

y\_second=all\_errorsK[1][0:5]

y\_third=all\_errorsK[2][0:5]

plt.plot(x\_vals, y\_first, color='r')

plt.plot(x\_vals, y\_second, color='y')

plt.plot(x\_vals, y\_third, color='g')

plt.legend ( ("h = 0.1", "h = 0.01","h = 0.0001") )

plt.show()

plt.savefig('graph6.png')

import sympy

def g(x, y):

xc, yc = 0, 0 # центр

R = 2 # радиус

return R\*\*2 - (x - xc)\*\*2 - (y - yc)\*\*2

r = sympy.symbols('r')

I\_exact = sympy.integrate(2\*sympy.pi\*r\*r, (r, 0, 2))

print ('Точное значение интеграла: ', I\_exact.evalf())

intervals=[(-2,2),(-2,2)]

monte\_int = Monte\_carlo\_method(intervals[:2], lambda x, y: np.sqrt(x\*\*2 + y\*\*2), g,1000)

print('Вычисленный:', monte\_int)

# Результат работы программы

h = 0.1 num= 10

Кратность Метод Монте-Карло Метод Квадратур Аналитически

1 0.531381083161214 0.500012469860719 0.5

2 65.56911143639249 55.96844163586613 58.18253870833333

3 1807.9652885179373 1750.484289852044 1819.73299572328

4 1074.3405853610645 1214.0866816515802 1262.1149348320193

5 455150.3324981824 2375057.0709809037 2469012.341265138

h = 0.01 num= 100

Кратность Метод Монте-Карло Метод Квадратур Аналитически

1 0.5114515156831498 0.5000000012911946 0.5

2 56.29449304047427 57.93143464960154 58.18253870833333

3 1623.0346517678574 1811.8793964960955 1819.73299572328

4 1181.016987755989 1256.6679023426018 1262.1149348320193

5 2862098.5760990544 2458356.583957715 2469012.341265138

h = 0.0001 num= 10000

Кратность Метод Монте-Карло Метод Квадратур Аналитически

1 0.501411642313671 0.5000000000000001 0.5

2 58.530573446130184 58.18253870833341 58.18253870833333

3 1817.2286342196255 1819.7329957232837 1819.73299572328

4 1263.9540747859871 1262.1149348320207 1262.1149348320193

5 2465965.4064612756 2469012.3412651415 2469012.341265138

Второй интеграл:

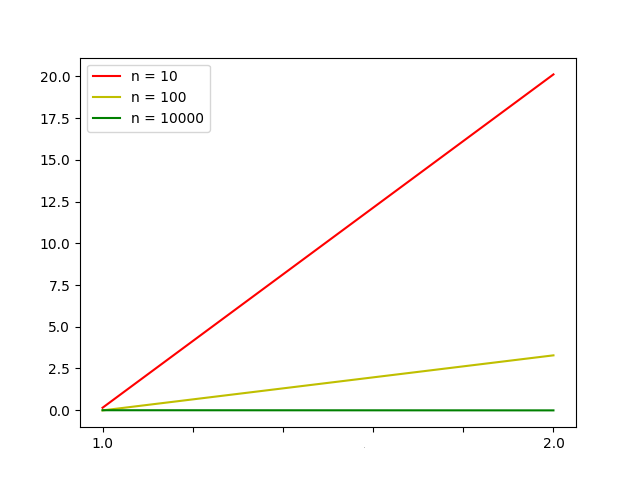
С помощью метода Монте-Карло 3.519646818291589

С помощью квадратур 3.620334371559588

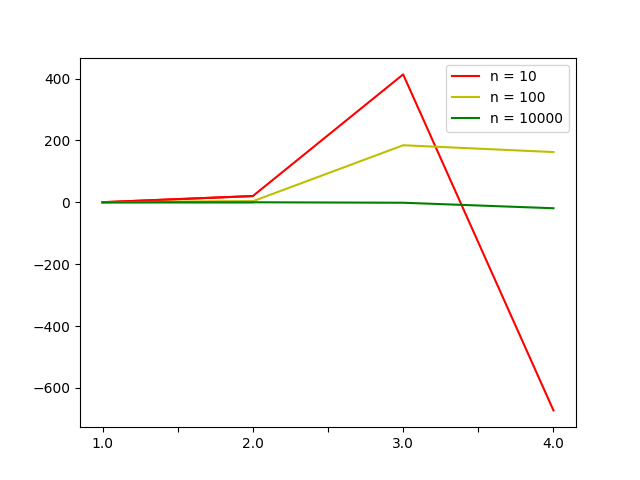
Вычисленный аналитически 3.5342917352885173

Графики зависимости погрешности от числа испытаний. (для ММК)

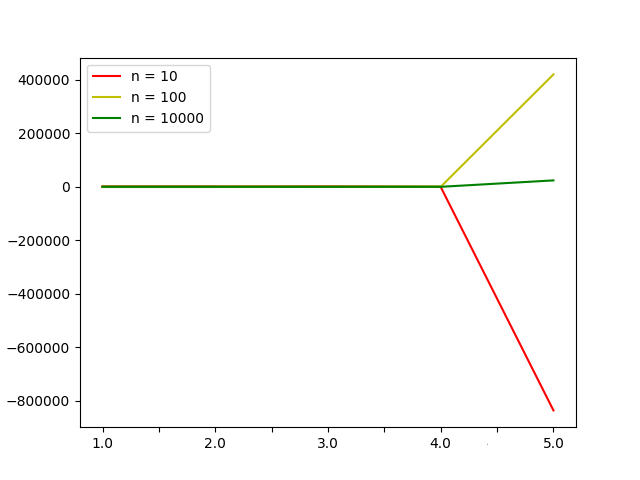
По оси X – кратность интеграла. По оси Y – погрешность.



Сравнение погрешностей для двойного интеграла.



Для четырехмерного.

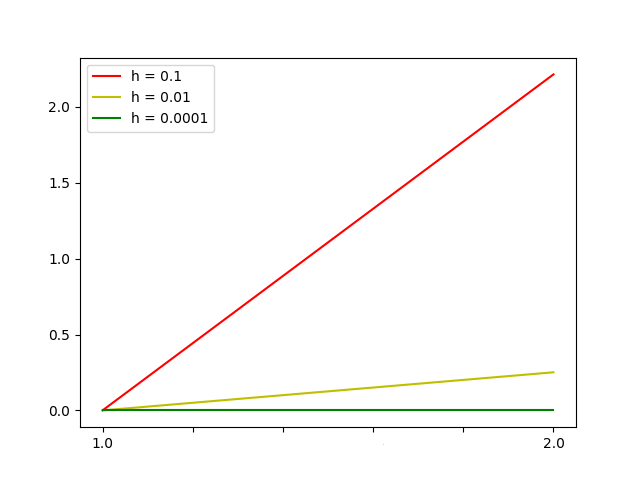


Для пятимерного.

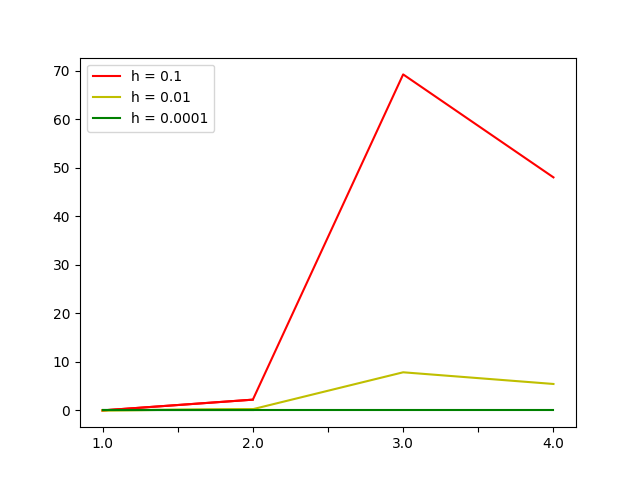
Легко заметить, что чем больше число испытаний n, тем точнее результаты вычисления.

Графики зависимости шага от числа испытаний. (для Метода Квадратур)

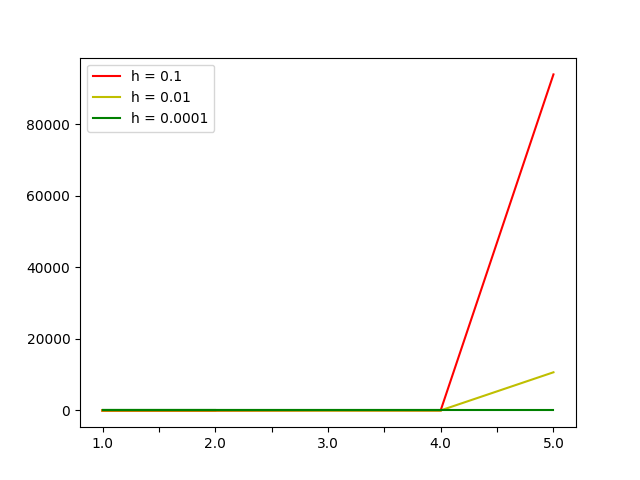
По оси X – кратность интеграла. По оси Y – шаг.



Сравнение погрешностей для двойного интеграла.



Для четырехмерного.



Для пятимерного.

Для метода квадратур все аналогично. Чем меньше шаг – тем меньше погрешность.

# Выводы

Выполнив данную курсовую работу, я научился применять метод Монте-Карло и квадратур для такой задачи, как вычисление многократных интегралов.

Работа была интересной, в ней применялись знания, полученные в течение всего изученного курса. Этот факт мне понравился, потому что пройденные методы интересно было куда-нибудь применить.

По окончанию данной работы я глубже разобрался в численных методах и их применении на практике. Они помогут мне в дальнейшем обучении.

# Литература

* Практическое применение численных методов на языке Python (http://slemeshevsky.github.io/)

### «[Численные методы», Формалев В.Ф., Ревизников Д.Л., 2006](http://nashol.com/2016011788006/chislennie-metodi-formalev-v-f-reviznikov-d-l-2006.html)

* Метод Симпсона (парабол).(http://www.cleverstudents.ru/integral/method\_of\_parabolas.html)