**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

**Лабораторная работа № 3**

по курсу «Численные методы».

Тема: «Приближений функций. Численные дифференцирование и интегрирование».

Студент: Шевчук П.В.

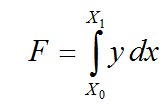
Группа: 80-308Б

Вариант: 13

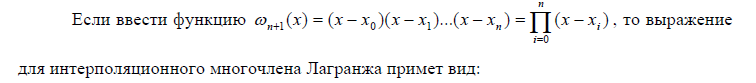
Оценка:

Москва, 2019

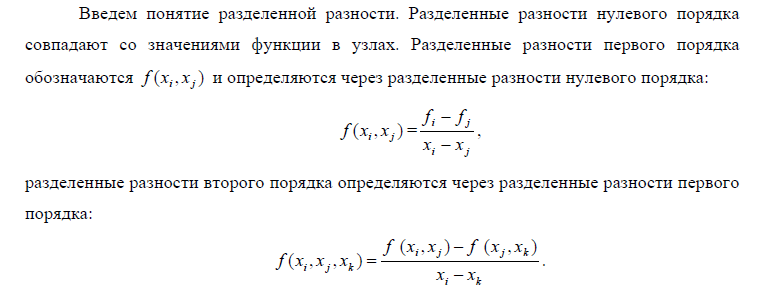
Постановка задачи.

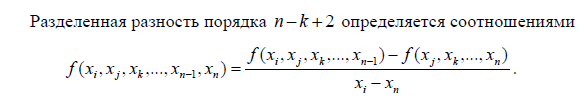
1. Используя таблицу значений  функции , вычисленных в точках  построить интерполяционные многочлены Лагранжа и Ньютона, проходящие через точки . Вычислить значение погрешности интерполяции в точке .
2. Построить кубический сплайн для функции, заданной в узлах интерполяции, предполагая, что сплайн имеет нулевую кривизну при  и . Вычислить значение функции в точке .
3. Для таблично заданной функции путем решения нормальной системы МНК найти приближающие многочлены a) 1-ой и б) 2-ой степени. Для каждого из приближающих многочленов вычислить сумму квадратов ошибок. Построить графики приближаемой функции и приближающих многочленов.
4. Вычислить первую и вторую производную от таблично заданной функции в точке .
5. Вычислить определённый интеграл, методами прямоугольников, трапеций, Симпсона с шагами . Оценить погрешность вычислений, используя метод Рунге-Ромберга:

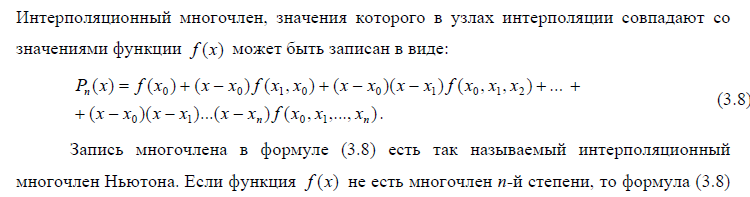
Описание методов.



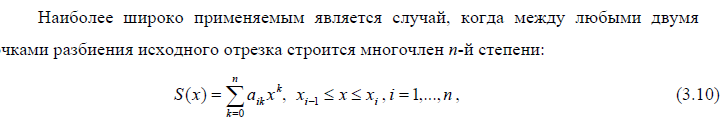


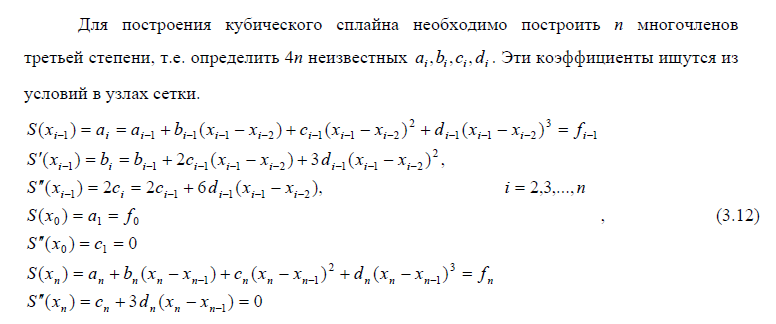




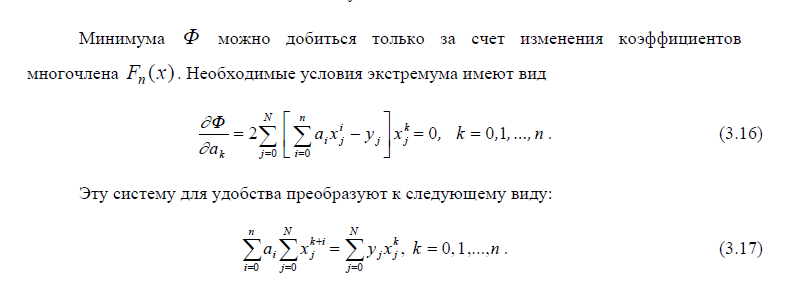
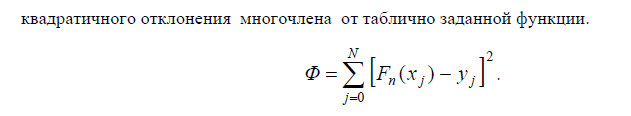
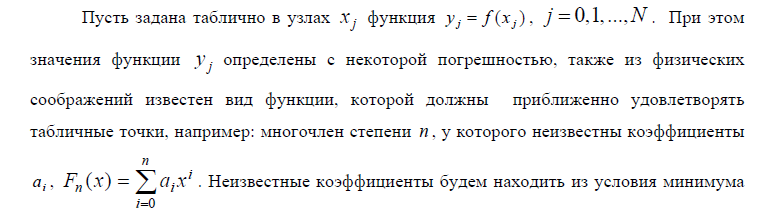


**Кубический сплайн** — гладкая функция, область определения которой разбита на конечное число отрезков, на каждом из которых она совпадает с некоторым **кубическим** многочленом (полиномом).

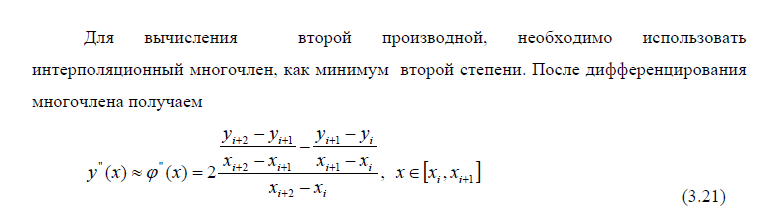
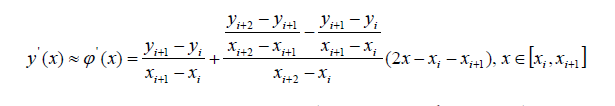
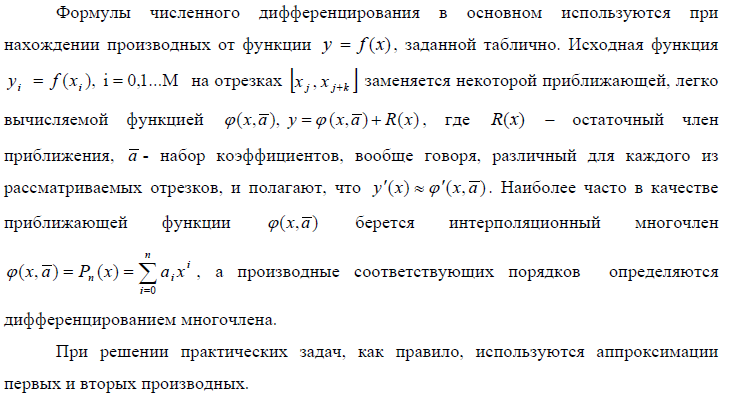




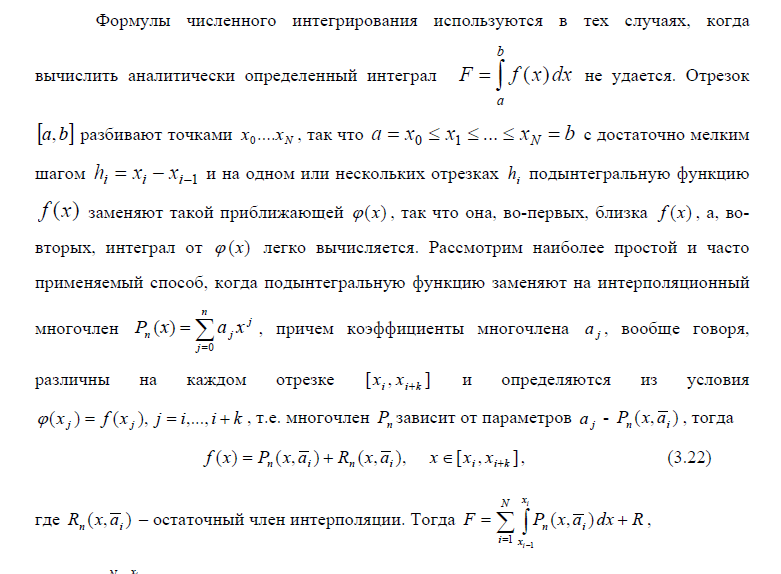
Метод наименьших квадратов.

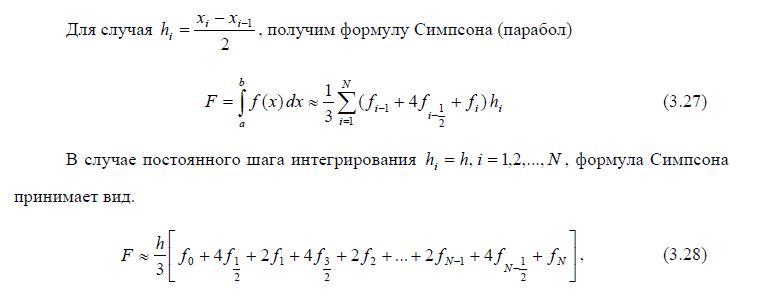
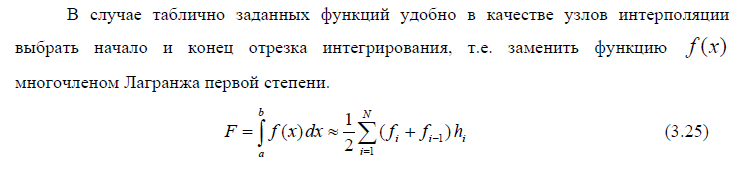
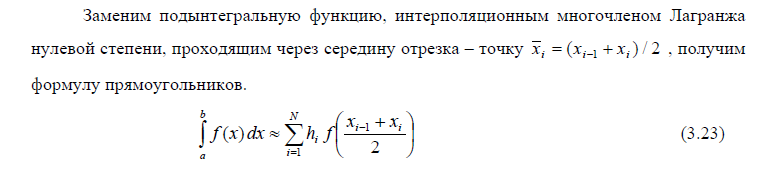


Численное дифференцирование



Численное интегрирование





Общая информация.

Данная работа состоит из пяти модулей, которые позволяют решать различные задачи решения СЛАУ и нахождения собственных векторов и собственных значений. Полученные в ходе расчетов результаты сохраняются в отдельный файл. Что касается технических деталей реализации, все программы написаны на языке python.

Запуск программы.

Чтобы воспользоваться программой, необходимо сделать всё аналогично.

Входные данные и результаты.

Вариант 1.

|  |  |
| --- | --- |
| **in1.txt**  0 0.166 0.3333 0.5  0.25 | **out1.txt**  Lagrange:  L(0.785) = 1.491160897361139  Eps = 0.00123  Newton:  N(0.785) = 1.4911608973611388  Eps = 0.00123  Orig:  cos(0.785) + 1 = 1.4923882691671997 |
| **In2.txt**  0.0 1.0 2.0 3.0 4.0  1.0 1.5403 1.5839 2.01 3.3464  1.5 | **Out2.txt**  x = [0.0, 1.0, 2.0, 3.0, 4.0]  y = [1.0, 1.5403, 1.5839, 2.01, 3.3464]  a = [1.0, 1.5403, 1.5839, 2.01]  b = [0.6844107142857142, 0.2520785714285716, 0.05897499999999983, 0.9211214285714284]  c = [0, -0.4323321428571427, 0.239228571428571, 0.6229178571428576]  d = [-0.14411071428571423, 0.22385357142857124, 0.12789642857142888, -0.2076392857142859]  f(1.5) = 1.5862 |
| **In3.txt**  -1.0 0.0 1.0 2.0 3.0 4.0  0.4597 1.0 1.5403 1.5839 2.010 3.3464 | **Out3.txt**  P\_1(x)=0.5002x+0.90641  P\_2(x)=0.06292x^2+0.31143x+0.86446  Error for ax + b: 0.48427  Error for ax^2 + bx + c: 0.33645 |
| **In4.txt**  0.2 0.5 0.8 1.1 1.4  12.906 5.5273 3.8777 3.2692 3.0319  0.8 | **Out4.txt**  f'(0.8) = -3.7635 центральный  f''(0.8) = 11.56778 2ая производная |
| **In5.txt** | **Out5.txt**  Method of rectangles = -0.1878310847553158 Step = 1.0  Method of rectangles = -0.19940628065257357 Step = 0.5  Error: 0.00386  ==========  Method of trapeziums = -0.23658183921341816 Step = 1.0  Method of trapeziums = -0.212206461984367 Step = 0.5  Error: 0.00813  ==========  Simpson's method = -0.2071717755928282 Step = 1.0  Simpson's method = -0.20408133624134991 Step = 0.5  Error: 0.00021 |

Выводы.

При выполнении данной лабораторной работы были освоены новые навыки программирования. Изучены новые алгоритмы.

Недостатком интерполяционного многочлена Лагранжа является необходимость полного пересчета всех коэффициентов в случае добавления дополнительных интерполяционных узлов. Чтобы избежать указанного недостатка используют интерполяционный многочлен в форме Ньютона.

Суть сплайн-интерполяции заключается в определении интерполирующей функции по формулам одного типа для различных непересекающихся промежутков и в стыковке значений функции и её производных на их границах.

Система (3.17) с увеличением степени n приближающего многочлена становится плохо обусловленной и решение её связано с большой потерей точности. Поэтому при использовании метода наименьших квадратов, как правило, используют приближающий многочлен не выше третьей степени.

При решении практических задач МНК, как правило, используются аппроксимации первых и вторых производных.

При использовании интерполяционных многочленов различной степени, получаются формулы численного интегрирования различного порядка точности.

Исходный код.

import argparse

import numpy as np

def lagrange(points):

def inner(x):

cnt\_points = len(points)

l = []

for i in range(cnt\_points):

l.append(np.prod([(x - points[j][0]) / (points[i][0] - points[j][0])

for j in range(cnt\_points) if i != j]))

return sum([points[idx][1] \* i for idx, i in enumerate(l)])

return inner

def newton(points):

def inner(x):

cnt\_points = len(points)

x\_point, y\_point = zip(\*points)

coef = [y\_point[0]]

for j, shift in zip(reversed(range(1, cnt\_points)), range(cnt\_points)):

tmp = []

for l, r in zip(range(j), range(1, j + 1)):

num1 = y\_point[l] - y\_point[r]

num2 = x\_point[l] - x\_point[r + shift]

tmp.append(num1 / num2)

y\_point = tmp

coef.append(y\_point[0])

res = 0

for i in range(cnt\_points):

res += coef[i] \* np.prod([x - x\_point[j] for j in range(i)])

return res

return inner

def get\_points(file\_name):

with open(file\_name) as f:

x = [float(num) \* np.pi for num in f.readline().split()]

points = [(i, np.cos(i)+i) for i in x]

x\_test = float(f.readline()) \* np.pi

return points, x\_test

def main():

parser = argparse.ArgumentParser()

parser.add\_argument('--input', required=True, help='Input test file')

parser.add\_argument('--output', required=True, help='File for answer')

args = parser.parse\_args()

points, x\_test = get\_points(args.input)

lg = lagrange(points)

nt = newton(points)

with open(args.output, 'w') as f:

tmp = round(x\_test, 3)

res1 = lg(tmp)

res2 = nt(tmp)

res3 = np.cos(tmp)+tmp

eps1 = abs(res1 - res3)

eps2 = abs(res2 - res3)

f.write(f'Lagrange:\nL({tmp}) = {res1}\nEps = {round(eps1, 5)}\n')

f.write(f'Newton:\nN({tmp}) = {res2}\nEps = {round(eps2, 5)}\n')

f.write(f'Orig:\ncos({tmp}) + 1 = {res3}\n')

if \_\_name\_\_ == "\_\_main\_\_":

main()

import argparse

import numpy as np

import matplotlib.pyplot as plt

def tma(matrix, d, shape):

a, b, c = zip(\*matrix)

p = [-c[0] / b[0]]

q = [d[0] / b[0]]

x = [0] \* (shape + 1)

for i in range(1, shape):

p.append(-c[i] / (b[i] + a[i] \* p[i - 1]))

q.append((d[i] - a[i] \* q[i - 1]) / (b[i] + a[i] \* p[i - 1]))

for i in reversed(range(shape)):

x[i] = p[i] \* x[i + 1] + q[i]

return x[:-1]

def spline(x, y):

size = len(x)

h = [x[i] - x[i - 1] for i in range(1, size)]

mtrx = [[0, 2 \* (h[0] + h[1]), h[1]]]

b = [3 \* ((y[2] - y[1]) / h[1] - (y[1] - y[0]) / h[0])]

for i in range(1, size - 3):

tmp = [h[i], 2 \* (h[i] + h[i + 1]), h[i + 1]]

mtrx.append(tmp)

b.append(3 \* ((y[i + 2] - y[i + 1]) / h[i + 1] - (y[i + 1] - y[i]) / h[i]))

mtrx.append([h[-2], 2 \* (h[-2] + h[-1]), 0])

b.append(3 \* ((y[-1] - y[-2]) / h[-1] - (y[-2] - y[-3]) / h[-2]))

c = tma(mtrx, b, size - 2)

a = []

b = []

d = []

c.insert(0, 0)

for i in range(1, size):

a.append(y[i - 1])

if i < size - 1:

d.append((c[i] - c[i - 1]) / (3 \* h[i - 1]))

b.append((y[i] - y[i - 1]) / h[i - 1] -

h[i - 1] \* (c[i] + 2 \* c[i - 1]) / 3)

b.append((y[-1] - y[-2]) / h[-1] - 2 \* h[-1] \* c[-1] / 3)

d.append(-c[-1] / (3 \* h[-1]))

return a, b, c, d

def get\_points(file\_name):

with open(file\_name) as f:

x = [float(num) for num in f.readline().split()]

y = [float(num) for num in f.readline().split()]

x\_test = float(f.readline())

return x, y, x\_test

def polyval(x0, x, k, coef):

a, b, c, d = coef

tmp = (x0 - x[k])

return a[k] + b[k] \* tmp + c[k] \* tmp\*\*2 + d[k] \* tmp\*\*3

def pol(x, x\_test, coef):

k = 0

for i, j in zip(x, x[1:]):

if i <= x\_test <= j:

break

k += 1

return polyval(x\_test, x, k, coef)

def main():

parser = argparse.ArgumentParser()

parser.add\_argument('--input', required=True, help='Input test file')

parser.add\_argument('--output', required=True, help='File for answer')

args = parser.parse\_args()

x, y, x\_test = get\_points(args.input)

coef = spline(x, y)

x1 = np.linspace(x[0], x[-1], 50)

y1 = [pol(x, i, coef) for i in x1]

plt.plot(x1, y1, color='b')

plt.scatter(x, y, color='r')

plt.show()

res = pol(x, x\_test, coef)

for i in x:

print(pol(x, i, coef))

with open(args.output, 'w') as f:

f.write(f'x = {x}\ny = {y}\n\n')

f.write(f'a = {coef[0]}\nb = {coef[1]}\nc = {coef[2]}\nd = {coef[3]}\n')

f.write(f'\nf({x\_test}) = {round(res, 4)}\n')

if \_\_name\_\_ == "\_\_main\_\_":

main()

import argparse

import numpy as np

import matplotlib.pyplot as plt

from numpy.linalg import inv

def get\_points(file\_name):

with open(file\_name) as f:

x = [float(num) for num in f.readline().split()]

y = [float(num) for num in f.readline().split()]

return x, y

def mls(k, x, y):

x = np.array(x)

t = np.array([[i\*\*j for i in x] for j in reversed(range(k))])

t\_trans = np.transpose(t)

g = t @ t\_trans

a = inv(g) @ t @ y

return a

def cal\_err(x, y, coef2, coef3):

y\_err2 = []

y\_err3 = []

for i in x:

y\_err2.append(np.polyval(coef2, i))

y\_err3.append(np.polyval(coef3, i))

err1 = sum([(y\_err2[idx] - i)\*\*2 for idx, i in enumerate(y)])

err2 = sum([(y\_err3[idx] - i)\*\*2 for idx, i in enumerate(y)])

return err1, err2

def main():

parser = argparse.ArgumentParser()

parser.add\_argument('--input', required=True, help='Input test file')

parser.add\_argument('--output', required=True, help='File for answer')

args = parser.parse\_args()

x, y = get\_points(args.input)

coef2 = mls(2, x, y)

coef3 = mls(3, x, y)

area = np.linspace(x[0], x[-1], 50)

y\_coef2 = [np.polyval(coef2, i) for i in area]

y\_coef3 = [np.polyval(coef3, i) for i in area]

plt.scatter(x, y, color='r')

plt.plot(area, y\_coef2, color='b')

plt.plot(area, y\_coef3, color='g')

plt.show()

with open(args.output, 'w') as f:

err1, err2 = cal\_err(x, y, coef2, coef3)

f.write(f'P\_1(x)={round(coef2[0], 5)}x+{round(coef2[1], 5)}\n')

f.write(f'P\_2(x)={round(coef3[0], 5)}x^2+{round(coef3[1], 5)}x+{round(coef3[2], 5)}\n\n')

f.write(f'Error for ax + b: {round(err1, 5)}\n')

f.write(f'Error for ax^2 + bx + c: {round(err2, 5)}\n')

if \_\_name\_\_ == "\_\_main\_\_":

main()

import argparse

def get\_points(file\_name):

with open(file\_name) as f:

x = [float(num) for num in f.readline().split()]

y = [float(num) for num in f.readline().split()]

x\_test = float(f.readline())

return x, y, x\_test

def first\_der(x, y, x0, k):

num1 = (y[k + 1] - y[k]) / (x[k + 1] - x[k])

num2 = (y[k + 2] - y[k + 1]) / (x[k + 2] - x[k + 1]) - num1

num2 = num2 / (x[k + 2] - x[k])

return num1 + num2 \* (2 \* x0 - x[k] - x[k + 1])

def second\_der(x, y, k):

num1 = (y[k + 2] - y[k + 1]) / (x[k + 2] - x[k + 1])

num2 = (y[k + 1] - y[k]) / (x[k + 1] - x[k])

return 2 \* (num1 - num2) / (x[k + 2] - x[k])

def main():

parser = argparse.ArgumentParser()

parser.add\_argument('--input', required=True, help='Input test file')

parser.add\_argument('--output', required=True, help='File for answer')

args = parser.parse\_args()

x, y, x\_test = get\_points(args.input)

k = 0

for i, j in zip(x, x[1:]):

if i <= x\_test <= j:

break

k += 1

res1 = first\_der(x, y, x\_test, k)

res2 = second\_der(x, y, k)

with open(args.output, 'w') as f:

f.write(f"f'({x\_test}) = {round(res1, 5)}\n")

f.write(f"f''({x\_test}) = {round(res2, 5)}\n")

if \_\_name\_\_ == "\_\_main\_\_":

main()

import numpy as np

import argparse

def f(x):

return x\*\*(2) / (x\*\*(3) - 27)

def simpson(a, b, h, n):

s = f(a) + f(b)

for i in range(1, n, 2):

s += 4 \* f(a + i \* h)

for i in range(2, n-1, 2):

s += 2 \* f(a + i \* h)

return s \* h / 3

def main():

parser = argparse.ArgumentParser()

parser.add\_argument('--output', required=True, help='File for answer')

args = parser.parse\_args()

a = -2

b = 2

h1 = 1.0

h2 = 0.5

x1 = np.linspace(a, b, int((b - a) / h1 + 1))

x2 = np.linspace(a, b, int((b - a) / h2 + 1))

y\_trap1 = [f(i) for i in x1]

y\_trap2 = [f(i) for i in x2]

rect1 = h1 \* sum([f((i + j) / 2) for i, j in zip(x1, x1[1:])])

rect2 = h2 \* sum([f((i + j) / 2) for i, j in zip(x2, x2[1:])])

trap1 = h1/2 \* sum([i + j for i, j in zip(y\_trap1[1:], y\_trap1)])

trap2 = h2/2 \* sum([i + j for i, j in zip(y\_trap2[1:], y\_trap2)])

simps1 = simpson(a, b, h1, int((b - a) / h1))

simps2 = simpson(a, b, h2, int((b - a) / h2))

with open(args.output, 'w') as f1:

f1.write(f'Method of rectangles = {rect1}\tStep = {h1}\n')

f1.write(f'Method of rectangles = {rect2}\tStep = {h2}\n')

f1.write(f'Error: {round(abs(rect1 - rect2) / 3, 5)}\n')

f1.write('='\*10)

f1.write(f'\nMethod of trapeziums = {trap1}\tStep = {h1}\n')

f1.write(f'Method of trapeziums = {trap2}\tStep = {h2}\n')

f1.write(f'Error: {round(abs(trap1 - trap2) / 3, 5)}\n')

f1.write('='\*10)

f1.write(f"\nSimpson's method = {simps1}\tStep = {h1}\n")

f1.write(f"Simpson's method = {simps2}\tStep = {h2}\n")

f1.write(f'Error: {round(abs(simps1 - simps2) / 15, 5)}\n')

'''

print(f'h = {h\_series[0]}\t{r}h = {h\_series[-1]}')

print('rectangles:')

print(rect, runge\_romberg(rect[0], rect[-1], r, n))

print("trapezium:")

print(trap, runge\_romberg(trap[0], trap[-1], r, n))

print("simpson:")

print(simp, runge\_romberg(simp[0], simp[-1], r, n))

print(f'analytical val = {wolfram\_val}')

'''

if \_\_name\_\_ == "\_\_main\_\_":

main()