Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №3 по курсу «Численные методы»

Студент:	П. П. Ветренко
Преподаватель:	И. Э. Иванов
Группа:	М8О-306Б
Вариант:	2
Дата:	
Оценка:	
Подпись:	

Москва, 2020

Лабораторная работа №3.1

Задача: Используя таблицу значений Y_i функции $y = f(x)$, вычисленных в точках $X_i, i = 0, \ldots, 3$ построить интерполяционные многочлены Лагранжа и Ньютона, проходящие через точки $\{X_i, Y_i\}$. Вычислить значение погрешности интерполяции в точке X^*.

$$y = cos(x), \qquad \text{a)} X_i = 0, \frac{\pi}{6}, \frac{\pi}{3}, \frac{\pi}{2}; \qquad \text{b)} X_i = 0, \frac{\pi}{6}, \frac{5\pi}{12}, \frac{\pi}{2}; \qquad X^* = \frac{\pi}{4}.$$

1 Описание метода решения

Многочлен Лагранжа
Интерполяционный многочлен Лагранжа имеет вид:

$$L_n(x) = \sum_{i=0}^{n} f_i \prod_{j=0, j\neq i}^{n} \frac{x - x_i}{x_i - x_j}.$$

Введём функцию

$$\omega_{n+1}(x) = \prod_{i=0}^{n}.$$

Тогда выражение для многочлена примет вид:

$$L_n(x) = \sum_{i=0}^{n} f_i \frac{\omega_{n+1}(x)}{(x - x_i)\omega'_{n+1}(x_i)}.$$

Для того, чтобы при добавлении новых интерполяционных узлов не приходилось пересчитывать все коэффициенты возьмём интерполяционный многочлен в форме Ньютона.

Многочлен Ньютона
Чтобы записать многочлен Ньютона, вводят такое понятие как разделённая разность. Её можно записать следующим образом:

$f(x_i, x_j) = \frac{f(x_i) - f(x_j)}{x_i - x_j}$ - разделённая разность первого порядка.

Разделённая разность второго порядка будет определяться через разности первого порядка:

$f(x_i, x_j, x_k) = \frac{f(x_i, x_j) - f(x_j, x_k)}{x_i - x_k}$

Тогда, при известных значениях $x_1, x_2, \dots, x_n$ функции $f(x)$, многочлен Ньютона можно записать как:

$P_n(x) = f(x_0) + (x - x_0) \cdot f(x_1, x_0) + (x - x_0)(x - x_1) \cdot f(x_0, x_1, x_2) + \cdots + (x - x_0)(x - x_1) \dots (x - x_n) \cdot f(x_0, x_1, \dots, x_n)$.

2 Входные и выходные данные

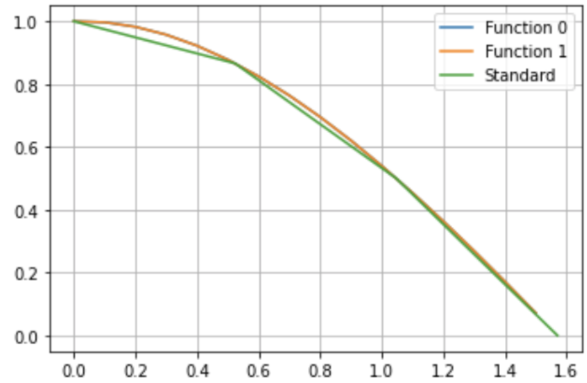
```
Лабораторная работа 3.1
Вариант 2

Многочлен Лагранжа
L(x) = - 1.161 + 3.017 (x + 0.0) (x - 1.0) (x - 1.6) - 1.742 (x + 0.0) (x - 0.5) (x - 1.6) + 0.000 (x +
0.5) (x - 1.0)
L(0.7853981633974483) =  0.706

Многочлен Ньютона
N(x) = 1.000 - 0.256 (x + 0.0) - 0.423 (x + 0.0) (x - 0.5) + 0.114 (x + 0.0) (x - 0.5) (x - 1.0)
N(0.7853981633974483) =  0.706


Функция
F(0.7853981633974483) =  0.707


Погрешность
|F(x) - L(x)| = 0.001217
|F(x) - N(x)| = 0.001217
```



3 Исходный код программы

```
import math
import numpy as np
import matplotlib.pyplot as plt

def f(x):
return math.cos(x)

def process(i,x):
n = len(x)
r = [1.0]
for j in range(n):
if j != i:
r.append(x[j])
r[0] /= x[i] -x[j]
return r

def lagrange(x):
n = len(x)
y = [f(val) for val in x]
L = []
for i in range(n):
li = process(i,x)
li[0] *= y[i]
L.append(li)
return L

def newton(x):
n = len(x)
y = [[f(val) for val in x]]
k = 1
N = [[y[0][0]]]
for i in range(n -1):
y.append([])
for j in range(len(y[i]) -1):
y[i + 1].append((y[i][j] -y[i][j + 1]) /
(x[j] -x[j + k]))
N.append([y[i+1][0]] + [x[k] for k in range(i + 1)])
k += 1
return N
```

```
def printer(P):
operator = False
for each in P:
if operator:
if each[0] <0:
print(f'-{abs(each[0]):5.3f}',end='')
else:
print(f'+ {abs(each[0]):5.3f}',end='')
for i in range(1,len(each)):
if each[i] >0:
print(f'(x -{each[i]:3.1f})',end='')
else:
print(f'(x + {abs(each[i]):3.1f})',end='')
else:
if each[0] <0:
print(f'-{abs(each[0]):5.3f}',end='')
else:
print(f'{abs(each[0]):5.3f}',end='')
operator = True
print('\n',end='')

def mm(lst):
r = 1
for i in lst:
r *= i
return r

def analyse(x,P):
func = 0
for each in P:
func += each[0] * \
mm(x -each[i] for i in range(1,len(each)))
return func

def show_plot(limit,x,y,step=0.1):
X = np.arange(x[0],x[-1],step)
Y = []
for i in range(len(limit)):
Y.append([limit[i](val) for val in X])
fif,ax = plt.subplots()
```

```
for i in range(len(Y)):
ax.plot(X,Y[i],label=f'Function {i}')
ax.plot(x,y,label='Standard')
ax.legend(loc='upper right')
ax.grid()
plt.show()

if __name__ == '__main__':
print('\nЛабораторная работа 3.1\nВариант 2')
x_1 = [0,math.pi/6,math.pi/3,math.pi/2]
x_2 = [0,math.pi/6,5*math.pi/12,math.pi/2]
x_check = math.pi/4

print('\nМногочлен Лагранжа')
L = lagrange(x_1)
print("L(x) =",end='')
printer(L)

Lx = analyse(x_check,L)
print(f'L({x_check}) = {Lx:6.3f}\n')

print('Многочлен Ньютона')
N = newton(x_1)
print("N(x) =",end='')
printer(N)

Nx = analyse(x_check,N)
print(f'N({x_check}) = {Nx:6.3f}\n')

print('\nФункция')
Fx = f(x_check)
print(f'F({x_check}) = {Fx:6.3f}\n')
print('\nПогрешность')
print(f'|F(x) -L(x)| = {abs(Fx -Lx):7.4}')
print(f'|F(x) -N(x)| = {abs(Fx -Nx):7.4}')

def l(x): return analyse(x,L)
def n(x): return analyse(x,N)
show_plot([l,n],x_1,[f(val) for val in x_1])
```

Лабораторная работа №3.2

Задача: Построить кубический сплайн для функции, заданной в узлах интерполяции, предполагая, что сплайн имеет нулевую кривизну при $x = x_0$ и $x = x_4$. Вычислить значение функции в точке $x = X^*$.

$X^* = 1.5;$
$x_1 = 0.0, \quad x_2 = 1.0, \quad x_3 = 2.0, \quad x_4 = 3.0, \quad x_5 = 4.0;$
$f_1 = 1.0, \quad f_2 = 0.86603, \quad f_3 = 0.5, \quad f_4 = 0.0, \quad f_5 = -0.5.$

1 Описание метода решения

Кубический сплайн
Кубический сплайн - такая гладкая функция, для построения которой необходимо найти кубический многочлен для каждого из данных отрезков. Каждый из n многочленов будет иметь вид:
$S(x) = a_i + b_i(x - x_{i-1}) + c_i(x - x_{i-1})^2 + d_i(x - x_{i-1})^3$,
где $x_{i-1} \leq x \leq x_i$, $i = 1, 2, \ldots, n$.
Для того, чтобы найти коэффициенты, необходимо сначала решить систему линейных алгебраических уравнений относительно одного из коэффициентов. При этом матрица системы будет трёхдиагональной. Затем, выразив из многочлена остальные коэффициенты, подстановкой уже известных данных мы сможем определить значения остальных коэффициентов.
Запишем систему относительно c_i:

$$\begin{cases} 4c_2 + c_3 = 1.19874 \\ c_2 + 4c_3 + c_4 = 3.25854 \\ c_3 + 4c_4 = 8.8575 \end{cases}$$

Остальные коэффициенты можно восстановить по формулам:

$$a_i = f_{i-1}, i = 1, \ldots, n; \quad b_i = \frac{(f_i - f_{i-1})}{h_i} - \frac{1}{3}h_i(c_{i+1} + 2c_i), \quad d_i = \frac{c_{i+1} - c_i}{3h_i}, i = 1, \ldots, n-1;$$

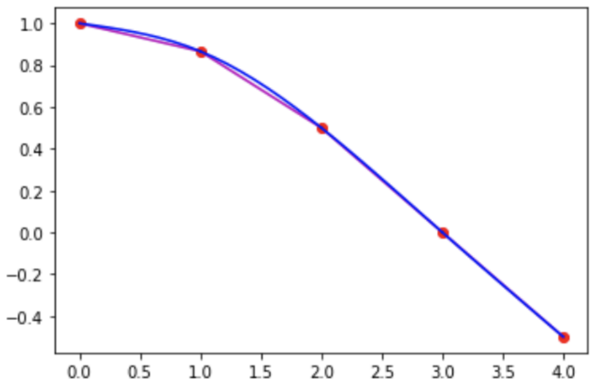
$$c_1 = 0, \quad b_n = \frac{(f_n - f_{n-1})}{h_n} - \frac{2}{3}h_n c_n, \quad d_n = -\frac{c_n}{3h_n}.$$

2 Входные и выходные данные

```
Лабораторная работа 3.2
Вариант 2:
X* = -0.5
i :       0       1       2       3       4
Xi:       0       1       2       3       4
Fi:       1    0.86603   0.5      0      -0.5


Начальная функция сплайна:
h is:  [0, 1, 1, 1, 1]
a is:  [0, 1, 0.86603, 0.5, 0]
b is:  [0, -0.0814, -0.2391, -0.462, -0.5109]
c is:  [0, 0, -0.15777, -0.0651, 0.01628]
d is:  [0, -0.05259, 0.03089, 0.02713, -0.00543]


Значение функции в точке X*:
F(1.5) = 0.7108987499999999
```



3 Исходный код программы

```
import math
import numpy as np
import matplotlib.pyplot as plt

target = """X* = -0.5
i :      0      1       2       3       4
Xi:      0      1       2       3       4
Fi:      1   0.86603   0.5      0      -0.5
"""

def get_h(x,i):
return x[i] -x[i -1]

def func(a,b,c,d,x):
return a + b * x + c * (x ** 2) + d * (x ** 3)

def get_a(f):
return [0] + [f[i] for i in range(len(f) -1)]

def get_b(f,c,h):
n = len(f) -1
b = [0]
for i in range(1,n):
b.append((f[i] -f[i -1]) / h[i] -1 / 3 * h[i] * (c[i + 1] + 2 * c[i]))
b.append((f[n] -f[n -1]) / h[n] -2 / 3 * h[n] * c[n])
return [round(i,4) for i in b]

def tma(a,b,c,d):
size = len(a)
p,q = [],[]
p.append(-c[0] / b[0])
q.append(d[0] / b[0])
for i in range(1,size):
p_tmp = -c[i] / (b[i] + a[i] * p[i -1])
q_tmp = (d[i] -a[i] * q[i -1]) / (b[i] + a[i] * p[i -1])
p.append(p_tmp)
q.append(q_tmp)
x = [0 for _ in range(size)]
x[size -1] = q[size -1]
```

```
for i in range(size -2,-1,-1):
x[i] = p[i] * x[i + 1] + q[i]
return x

def get_c(f,h):
n = len(f)
a = [0] + [h[i -1] for i in range(3,n)]
b = [2 * (h[i -1] + h[i]) for i in range(2,n)]
c = [h[i] for i in range(2,n -1)] + [0]
d = [3 * ((f[i] -f[i -1]) / h[i] -((f[i -1] -f[i -2]) / h[i -1])) for i in
range(2,n)]
x = tma(a,b,c,d)
res = [0,0] + [round(i,5) for i in x]
return res

def get_d(h,c):
n = len(c) -1
d = [0]
for i in range(1,n):
d.append((c[i + 1] -c[i]) / (3 * h[i]))
d.append(-c[n] / (3 * h[n]))
return [round(i,5) for i in d]

def get_interval(x,x_check):
for i in range(len(x) -1):
if x[i] <= x_check <= x[i + 1]:
return i

def spline(x,f,x_check):
n = len(x)
h = [0]
for i in range(1,n):
h.append(get_h(x,i))
a = get_a(f)
c = get_c(f,h)
b = get_b(f,c,h)
d = get_d(h,c)
print('h is: ',h)
print('a is: ',a)
print('b is: ',b)
print('c is: ',c)
```

```
print('d is: ',d,'\n')
tmp = get_interval(x,x_check)
ans = func(a[tmp + 1],b[tmp + 1],c[tmp + 1],d[tmp + 1],x_check -x[tmp])
return ans,a,b,c,d

def show_plot(x,f,a,b,c,d):
X,Y = [],[]
for i in range(len(x) -1):
x_i = np.linspace(x[i],x[i + 1],10,endpoint=True)
y_i = [func(a[i + 1],b[i + 1],c[i + 1],d[i + 1],j -x[i]) for j in x_i]
X.append(x_i)
Y.append(y_i)
fig,ax = plt.subplots()
ax.scatter(x,f,color='r')
ax.plot(x,f,color='m')
for i in range(len(x) -1):
ax.plot(X[i],Y[i],color='b')
plt.savefig('spline.png')
plt.show()

if __name__ == '__main__':
print('\nЛабораторная работа 3.2\nВариант 2:')
print(target)
x_i = [0,1,2,3,4]
f_i = [1,0.86603,0.5,0,-0.5]
x_check = 1.5

print('\nНачальная функция сплайна:')
y,a,b,c,d = spline(x_i,f_i,x_check)

print('\nЗначение функции в точке X*:')
print(f'F({x_check}) = {y}')
show_plot(x_i,f_i,a,b,c,d)
```

Лабораторная работа №3.3

Задача: Для таблично заданной функции путем решения нормальной системы МНК найти приближающие многочлены а) 1-ой и б) 2-ой степени. Для каждого из приближающих многочленов вычислить сумму квадратов ошибок. Построить графики приближаемой функции и приближающих многочленов.

$x_0 = -1.0, \quad x_1 = 0.0, \quad x_2 = 1.0, \quad x_3 = 2.0, \quad x_4 = 3.0, \quad x_5 = 4.0;$
$f_0 = 0.86603, \quad f_1 = 1.0, \quad f_2 = 0.86603, \quad f_3 = 0.5, \quad f_4 = 0.0, \quad f_5 = -0.5.$

1 Описание метода решения

Метод наименьших квадратов
Метод наименьших квадратов (МНК) - метод, позволяющий решать переопределённые системы уравнений и аппроксимировать точечные значения функции. Поэтому для того, чтобы построить приближающие многочлены, мы воспользуемся системой метода наименьших квадратов. Она имеет следующий вид:

$$\sum_{i=0}^{n} a_i \sum_{j=0}^{N} x_j^{k+i} = \sum_{j=0}^{N} y_j x_j^k,$$

где $k = 0, 1, \ldots, n, \quad x_j$ - это узлы, $y_j = f(x_j), \quad j = 0, 1, \ldots, N$.
Неизвестные коэффициенты $a, b, c, \ldots$ будем искать с помощью нормальной системы МНК. Например для многочлена $ax + b$ первой степени будем искать так:

$$\begin{cases} 6b - 3a = 11.66041 \\ -3b + 19a = 16.70857 \end{cases}$$

Для приближающего многочлена второй степени:

$$\begin{cases} 6c - 3b + 19a = 11.66041 \\ -3c + 19b - 27a = 16.70857 \\ 19c - 27b + 115a = 37.66504 \end{cases}$$

Сумму квадратов ошибок найдём по формуле:

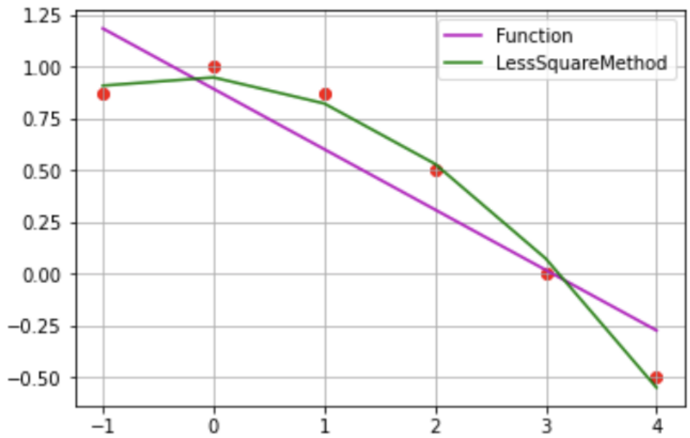
$$\Phi = \sum_{j=0}^{N} [F_n(x_j) - y_j]^2.$$

2 Входные и выходные данные

```
Лабораторная работа 3.3
Вариант 2:

i:        0         1         2        3        4        5
Xi:     -1.0       0.0       1.0      2.0      3.0      4.0
Yi:    0.86603    1.0     0.86603    0.5      0.0     -0.5

Степень = 1
F1(x) = 0.89232x^0 + -0.29132x^1
Степень = 2
F2(x) = 0.94749x^0 + -0.04307x^1 + -0.08275x^2
```




```
Погрешность функции F1 = 0.27082
Погрешность функции F2 = 0.01518
```

3 Исходный код программы

```
import matplotlib.pyplot as plt
import scipy.linalg as la

target = '''
i:        0        1        2       3       4       5
Xi:     -1.0      0.0      1.0     2.0     3.0     4.0
Yi:    0.86603    1.0    0.86603   0.5     0.0    -0.5
'''

def func(x,values):
return sum([c * (x ** i) for i,c in enumerate(values)])

def sse(f,y):
return round(sum([(f_i -y_i) ** 2 for f_i,y_i in zip(f,y)]),5)

def mls(n,x,y):
matrix = [[] for _ in range(n + 1)]
size = len(matrix)
for i in range(n + 1):
for j in range(n + 1):
matrix[i].append(sum([x_j ** (i + j) for x_j in x]))
b = [0 for _ in range(n + 1)]
for i in range(n + 1):
b[i] = sum([y_j * (x_j ** i) for x_j,y_j in zip(x,y)])
(P,L,U) = la.lu(matrix)
new_b = la.solve(matrix,b)
return [round(i,5) for i in new_b]

def f_printer(coefs):
n = len(coefs)
f = f'F{n -1}(x) = '
for i in range(n):
f += f'{coefs[i]}x^{i} + '
f = f[:-2]
return f

if __name__ == '__main__':
print('\nЛабораторная работа 3.3\nВариант 2:')
print(target)
```

```
x = [-1.0,0.0,1.0,2.0,3.0,4.0]
y = [0.86603,1.0,0.86603,0.5,0.0,-0.5]
F = []
err = []
coefs = []

for degree in [1,2]:
print(f'Степень = {degree}')
coefs.append(mls(degree,x,y))
print(f_printer(coefs[degree -1]))
F.append([func(i,coefs[degree -1]) for i in x])
err.append(sse(F[degree -1],y))

plt.scatter(x,y,color='r')
plt.plot(x,F[0],color='m',label='Function')
plt.plot(x,F[1],color='g',label='LessSquareMethod')
plt.legend(loc='best')
plt.grid()
plt.savefig('3_3.png')
plt.show()

k = 1
for i in err:
print(f'Погрешность функции F{k} = {i}')
k += 1
```

Лабораторная работа №3.4

Задача: Вычислить первую и вторую производную от таблично заданной функции $y_i = f(x_)$, $i = 0, 1, 2, 3, 4$, в точке $x = X^*$.

$X^* = 1.0$
$x_0 = -1.0, \quad x_1 = 0.0, \quad x_2 = 1.0, \quad x_3 = 2.0, \quad x_4 = 3.0;$
$y_0 = -0.5, \quad y_2 = 0.0, \quad y_2 = 0.5, \quad y_3 = 0.86603, \quad y_4 = 1.0.$

1 Описание метода решения

Функцию $y = f(x)$ заменим на приближающую функцию $y = \varphi(x, \overline{a}) + R(x)$, где $R(x)$ - остаточный член приближения, а $\overline{a}$ - уникальный набор коэффициентов для каждого из отрезков. Так как $y^{'}(x) \approx \varphi^{'}(x, \overline{a})$, найдя производную для функции φ мы сможем определить $y^{'}$.
По отрезкам аппроксимируем таблично заданную функцию. В таком случае первую производную найдём по формуле:

$$y^{'} = \varphi^{'}(x) = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}.$$

Вторую производную ищем следующим образом:

$$y^{''} = \varphi^{''}(x) = 2 \cdot \frac{\frac{y_{i+2} - yi+1}{x_{i+2} - x_{i+1}} - \frac{y_{i+1} - yi}{x_{i+1} - x_i}}{x_{i+2} - x_i}.$$

Так как точка X^* совпадает с правой границей отрезка $[x_2, x_3]$, то можем найти левостороннюю производную по формуле:

$$y^{'} = \frac{y_3 - y_2}{x_3 - x_2}$$

Аналогично можно найти правостороннюю производную:

$$y^{'} = \frac{y_4 - y_3}{x_4 - x_3}$$

2 Входные и выходные данные

```
Лабораторная работа 3.4
Вариант 2:
X* = 1.0
i:        0        1        2        3        4
Xi:     -1.0      0.0      1.0      2.0      3.0
Yi:     -0.5      0.0      0.5    0.86603    1.0

Первая производная = 0.433015
Вторая производня = -0.13397
```

3 Исходный код программы

```
target = '''X* = 1.0
i:         0        1        2         3        4
Xi:      -1.0      0.0      1.0       2.0      3.0
Yi:      -0.5      0.0      0.5     0.86603    1.0
'''

def find_interval(x_i,x0):
for i in range(len(x_i) -1):
if x_i[i] <= x0 <= x_i[i + 1]:
return i

def first_derivative(x_i,y_i,x0):
i = find_interval(x_i,x0)
left = (y_i[i + 1] -y_i[i]) / (x_i[i + 1] -x_i[i])
right = ((y_i[i + 2] -y_i[i + 1]) / (x_i[i + 2] -x_i[i + 1]) -left) / \
(x_i[i + 2] -x_i[i]) * (2 * x0 -x_i[i] -x_i[i + 1])
return left + right

def second_derivative(x_i,y_i,x0):
i = find_interval(x_i,x0)
left = (y_i[i + 1] -y_i[i]) / (x_i[i + 1] -x_i[i])
right = 2 * ((y_i[i + 2] -y_i[i + 1]) / (x_i[i + 2] -x_i[i + 1]) -left) / \
(x_i[i + 2] -x_i[i])
return right

if __name__ == '__main__':
print('\nЛабораторная работа 3.4\nВариант 2:')
print(target)
x0 = 1.0
x_i = [-1.0,0.0,1.0,2.0,3.0]
y_i = [-0.5,0.0,0.5,0.86603,1.0]
print(f'Первая производная = {round(first_derivative(x_i,y_i,x0),6)}')
print(f'Вторая производня = {round(second_derivative(x_i,y_i,x0),6)}')
```

Лабораторная работа №3.5

Задача: Вычислить определенный интеграл $F = \int_{x_0}^{x_1} y dx$ методами прямоугольников, трапеций, Симпсона с шагами h_1, h_2. Оценить погрешность вычислений, используя Метод Рунге-Ромберга:

$$y = \frac{x}{(3x+4)^2},$$
$$X_0 = 0, \quad X_k = 4, \quad h_1 = 1, \quad h_2 = 0.5.$$

1 Описание метода решения

В случае, если не удаётся вычислить интеграл $F = \int_{x_0}^{x_1} ydx$ необходимо заменить функцию $y = f(x)$, на приближающую функцию $\varphi(x)$, для которой проще найти интеграл. На отрезке $[x_0, x_1]$ с небольшим шагом h_i заменим функцию $y = f(x)$ на интерполяционный многочлен Лагранжа нулевой степени, проходящий через середину отрезка, чтобы вычислить интеграл по формуле прямоугольников, которая имеет следующий вид:

$$\int_{x_0}^{x_1} ydx \approx \sum_{i=1}^{N} h_i f(\frac{x_{i-1} + x_i}{2}).$$

Заменив первоначальную функцию на многочлен Лагранжа первой степени, который проходит через концы отрезка, мы получим формулу трапеций:

$$\int_{x_0}^{x_1} ydx \approx \frac{1}{2} \sum_{i=1}^{N} (f_i + f_{i-1}) h_i.$$

Если заменить подынтегральную функцию интерполяционным многочленом Лагранжа второй степени, проходящим через концы и середину отрезка, то мы получим формулу Симпсона:

$$\int_{x_0}^{x_1} ydx \approx \frac{1}{3} \sum_{i=1}^{N} (f_{i-1} + 4f_{i-\frac{1}{2}} + f_i) h_i.$$

Если есть результаты вычисления определенного интеграла на сетке с шагом h и на сетке с шагом kh, то можно использовать формулу Рунге-Ромберга:

$$\int_{x_0}^{x_1} ydx = F_h + \frac{F_h - F_{kh}}{k^p - 1} + O(h^{p+1}).$$

2 Входные и выходные данные

```
Лабораторная работа 3.5
Вариант 2:

Текущий h = 1
x = [0, 1, 2, 3, 4]
y = [0.0, 0.02040816326530612, 0.02, 0.01775147928994083, 0.015625]

Метод прямоугольников:
Value = 0.07284061196629331

Метод трапеций:
Value = 0.04822066326530612

Метод Симпсона:
Value = 0.0694211900736626


Текущий h = 0.5
x = [0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0]
y = [0.0, 0.01652892561983471, 0.02040816326530612, 0.020761245674740483, 0.02, 0.01890359168241966, 0.01775147928
994083, 0.016646848989298454, 0.015625]

Метод прямоугольников:
Value = 0.0713276669809035

Метод трапеций:
Value = 0.0610829527661209

Метод Симпсона:
Value = 0.07055112216261118

Погрешности:
Значение интеграла: 0.06765391396987852
Значение методом прямоугольников: 0.07082335198577355, погрешность: 0.003169438015895032
Значение методом трапеций:        0.06537038259972583, погрешность: 0.0022835313701526916
Значение методом Симпсона:        0.07062645096854109, погрешность: 0.002972536998662567
```

3 Исходный код программы

```
import numpy as np

target = '''
y = x / (3 * x + 4)^2
x0 = 0
x_k = 4
h1 = 1
h2 = 0.5
'''

def func(x):
return x/(3 * x + 4)**2

def get_x(x0,x,step):
return [i for i in np.arange(x0,x + step,step)]

def get_y(x):
return [func(i) for i in x]

def rectangle(x,h):
return h * sum([func((x[i] + x[i + 1]) / 2) for i in range(len(x) -1)])

def trapeze(x,h):
y = get_y(x)
return h * (y[0] / 2 + sum([y[i] for i in range(1,len(y) -2)]) + y[len(y) -1]
/ 2)

def simpson(x,h):
y = get_y(x)
return h / 3 * (y[0] +
sum([4 * y[i] for i in range(1,len(y) -1,2)]) +
sum([2 * y[i] for i in range(2,len(y) -2,2)]) +
y[len(y) -1])

def runge_Romberg(res,true_value):
k = res[1]['h'] / res[0]['h']
val_rec = [res[0]['rec'] + (res[0]['rec'] -res[1]['rec']) / (k ** 2 -1),
abs(res[0]['rec'] + (res[0]['rec'] -res[1]['rec']) / (k ** 2 -1) -true_value)]
```

```
val_trp = [res[0]['trp'] + (res[0]['trp'] -res[1]['trp']) / (k ** 2 -1),
abs(res[0]['trp'] + (res[0]['trp'] -res[1]['trp']) / (k ** 2 -1) -true_value)]

val_smp = [res[0]['smp'] + (res[0]['smp'] -res[1]['smp']) / (k ** 4 -1),
abs(res[0]['smp'] + (res[0]['smp'] -res[1]['smp']) / (k ** 4 -1) -true_value)]
return {'rec': val_rec,'trp': val_trp,'smp': val_smp}

if __name__ == '__main__':
print('\nЛабораторная работа 3.5\nВариант 2:')
x0 = 0
x = 4
h = [1,0.5]
true_value = 0.06765391396987852
res = []
for h_i in h:
print('\nТекущий h =',h_i)
X = get_x(x0,x,h_i)
print(f'x = {X}')
y = get_y(X)
print(f'y = {y}')

print('\nМетод прямоугольников:')
res_rec = rectangle(X,h_i)
print(f'Value = {res_rec}')

print('\nМетод трапеций:')
res_trp = trapeze(X,h_i)
print(f'Value = {res_trp}')

print('\nМетод Симпсона:')
res_smp = simpson(X,h_i)
print(f'Value = {res_smp}')
print()

res.append({"h": h_i,
"rec": res_rec,
"trp": res_trp,
"smp": res_smp})

err = runge_Romberg(res,true_value)
```

```
print('Погрешности:')
print(f'Значение интеграла: {true_value}')
print('Значение методом прямоугольников: {},погрешность:
{}'.format(err['rec'][0],err['rec'][1]))
print('Значение методом трапеций:        {},погрешность:
{}'.format(err['trp'][0],err['trp'][1]))
print('Значение методом Симпсона:        {},погрешность:
{}'.format(err['smp'][0],err['smp'][1]))
```

4 Выводы

Мне довелось поработать с графическими методами языка Python для того, чтобы построить графические изображения.
Также я научилась реализовывать новые методы в программном коде, для приближения функций и численного дифференцирования и интегрирования. Полученные навыки однозначно пригодятся в дальнейшем, для решения математических и вычислительных задач.