Московский авиационный институт
(Национальный исследовательский университет)
Факультет прикладной математики и физики
Кафедра вычислительной математики и программирования

Лабораторная работа № 3

по курсу «Численные методы»

Тема: численные методы решения СЛАУ.

Студент: Сыроежкин К.Г.
Группа: 80-304б
Преподаватель: Гидаспов В.Ю.
Оценка:

Москва, 2021

Задание 1

1) Постановка задачи:

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

Вариант 16:

16. $y = \ln(x) + x$, а) $X_i = 0.1, 0.5, 0.9, 1.3$; б) $X_i = 0.1, 0.5, 1.1, 1.3$; $X^* = 0.8$.

2) Теория:

Интерполяционный многочлен Лагранжа имеет вид

$$\omega_{n+1}(x) = (x - x_0)(x - x_1)...(x - x_n) = \prod_{i=0}^{n}(x - x_i)$$

Если ввести функцию

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

то многочлен примет вид

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

Введем понятие разделенной разности. Разделенная разность нулевого порядка совпадает со значением ф-ции в узле.
Разделенная разность первого порядка:

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

Разделенная разность второго порядка:

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

Разделенная разность n – k + 2 порядка:

$$f(x_i, x_j, x_k, \ldots, x_{n-1}, x_n) = \frac{f(x_i, x_j, x_k, \ldots, x_{n-1}) - f(x_j, x_k, \ldots, x_n)}{x_i - x_n}.$$

Многочлен Ньютона пишется в виде:

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

3) Полученный ответ:

```
Первый набор значений:
Ланграж:
X =  [0.1 0.5 0.9 1.3]
Y =  [-2.20258509 -0.19314718  0.79463948  1.56236426]
L(x) = 5.735898679671992(x - 0.5)(x - 0.9)(x - 1.3)-1.5089623481245722(x -
0.1)(x - 0.9)(x - 1.3)-6.208120971423232(x - 0.1)(x - 0.5)(x -
1.3)+4.0686569387174245(x - 0.1)(x - 0.5)(x - 0.9)
L(0.8) = 0.5996357042970932
delta =  0.022779255611302895
Ньютон:
N(x) = -2.2025850929940454+5.02359478108525(x - 0.1)-3.1926601485374406(x -
0.5)(x - 0.1)+2.087472298841613(x - 0.9)(x - 0.5)(x - 0.1)
L(0.8) = 0.5996357042970936
delta =  0.02277925561130323

Второй набор значений:
Ланграж:
X =  [0.1 0.5 1.1 1.3]
Y =  [-2.20258509 -0.19314718  1.19531018  1.56236426]
L(x) = 4.588718943737595(x - 0.5)(x - 1.1)(x - 1.3)-1.0059748987497148(x -
0.1)(x - 1.1)(x - 1.3)-9.960918165036043(x - 0.1)(x - 0.5)(x -
1.3)+8.13731387743485(x - 0.1)(x - 0.5)(x - 1.1)
L(0.8) = 0.6341106211498606
delta =  0.057254172464070274
Ньютон:
N(x) = -2.2025850929940454+5.02359478108525(x - 0.1)-2.709499180478133(x -
0.5)(x - 0.1)+1.7591397573866874(x - 1.1)(x - 0.5)(x - 0.1)
L(0.8) = 0.6341106211498606
delta =  0.057254172464070274
```

4) Код программы:

```
import numpy as np
```

```
def f(x):
    return np.log(x)+x

def get_X(file):
    mat = []
    answ = []
    with open(file) as file_handler:
        for line in file_handler:
            row = list(map(float, line.split(',')))
            mat.append(row)
    return np.array(mat)


def get_w(X,i):
    res = 1
    for k in range(len(X)):
        if k != i:
            res *= X[i] -X[k]
    return res

def get_wx(x,X):
    res = 1
    n = len(X)
    for i in range(n):
        res*= x-X[i]
    return res

def get_coefs_langrange(X,Y,i):
    return Y[i]/get_w(X,i)


def langrange(x, X, Y):
    n = len(X)
    L = 0
    coefs = []
    for i in range(n):
        coef = get_coefs_langrange(X,Y,i)
        coefs.append(coef)
        L += get_wx(x,X)*Y[i]/(get_w(X,i)*(x - X[i]))
    return coefs,L


def pretty_langrange(x,X,Y):
    L_str = "L(x) = "
    coefs, L = langrange(x,X,Y)
    for i in range(len(coefs)):
        if coefs[i] > 0 and i > 0:
            L_str +="+" + str(coefs[i])
        else:
            L_str += str(coefs[i])
        for k in range(len(X)):
```

```
            if k != i:
                if (X[k]>0):
                    L_str+="(x - {})".format(X[k])
                else:
                    L_str+="(x + {})".format(-X[k])
    return L_str, L


def get_newton_coef(n, i, j, X, Y):
    if(n==0):
        return (Y[i] - Y[j]) / (X[i] - X[j])
    else:
        return (get_newton_coef(n-1, i, j-1, X, Y) - get_newton_coef(n-1, i+1,
j, X, Y)) / (X[i] - X[j])

def get_w_newton(x,n, X):
    res = 1
    for i in range(n+1):
        res*= x-X[i]
    return res


def newton_polynom(x, X, Y):
    n = len(X)
    N = Y[0] + (x- X[0])*get_newton_coef(0, 1, 0, X, Y)
    coefs = [get_newton_coef(0, 1, 0, X, Y)]
    for i in range(1, n-1):
        tmp = get_newton_coef(i, 0, i+1, X, Y)
        coefs.append(tmp)
        N += get_w_newton(x, i, X)*tmp
    return coefs, N

def pretty_newton(x, X, Y):
    coefs, N = newton_polynom(x,X,Y)
    N_str = "N(x) = {}".format(Y[0])
    for i in range(0,len(coefs)):
        if coefs[i] > 0:
            N_str += "+"
        N_str += str(coefs[i])
        for j in np.arange(i,-1,-1):
            if (X[j]>0):
                N_str+="(x - {})".format(X[j])
            else:
                N_str+="(x + {})".format(-X[j])
    return N_str, N


print("\nПервый набор значений:")
print("Ланграж:")
X1 = get_X("test.txt")[0]
X2 = get_X("test.txt")[1]
```

```
X_ = 0.8
Y1 = f(X1)
Y2 = f(X2)
langrange(X_,X2,Y2)
L_str, L_value = pretty_langrange(X_,X1,Y1)
print("X = ", X1)
print("Y = ", Y1)
print(L_str)
print("L({}) = {}".format(X_,L_value))
print("delta = ", abs(L_value - f(X_)))
print("Ньютон:")
N_str, N_value = pretty_newton(X_,X1,Y1)
print(N_str)
print("L({}) = {}".format(X_,N_value))
print("delta = ", abs(N_value - f(X_)))

print("\nВторой набор значений:")
print("Ланграж:")
L_str, L_value = pretty_langrange(X_,X2,Y2)
print("X = ", X2)
print("Y = ", Y2)
print(L_str)
print("L({}) = {}".format(X_,L_value))
print("delta = ", abs(L_value - f(X_)))
print("Ньютон:")
N_str, N_value = pretty_newton(X_,X2,Y2)
print(N_str)
print("L({}) = {}".format(X_,N_value))
print("delta = ", abs(N_value - f(X_)))
```

Задание 2

1) Постановка задачи:

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 $x = X^*$.

Вариант 16:

16. $X^* = 0.8$

i	0	1	2	3	4
x_i	0.1	0.5	0.9	1.3	1.7
f_i	-2.2026	-0.19315	0.79464	1.5624	2.2306

2) Теория:

Для построения кубического сплайна необходимо построить многочлен третьей степени для каждого отрезка разбиения, всего n многочленов. Сам многочлен представляется в виде:

$$S(x) = a_i + b_i(x - x_{i-1}) + c_i(x - x_{i-1})^2 + d_i(x - x_{i-1})^3,$$
$$x_{i-1} \le x \le x_i,\ i = 1,2,\ldots,n.$$

Коэффициент c находится при помощи решения системы уравнений, задающих условия в узлах сетки. Остальные коэффициенты ищутся по формулам:

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

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

3) Полученный ответ:

```
-2.2026 + 5.672925892857142(x - 0.1) + 0.0(x - 0.1)^2 + -4.05813058035714(x
- 0.1)^3
-0.19315 + 3.7250232142857134(x - 0.5) + -4.86975669642857(x - 0.5)^2 +
4.327215401785712(x - 0.5)^3 V
0.79464 + 1.90628125(x - 0.9) + 0.3229017857142859(x - 0.9)^2 + -
0.7252622767857143(x - 0.9)^3
1.5624 + 1.8164767857142854(x - 1.3) + -0.5474129464285713(x - 1.3)^2 +
0.45617745535714266(x - 1.3)^3

0.6029136774553573
```

4) Код программы:

```
def get_PQ(mat,answ):
    P1 = -mat[0,1]/mat[0,0]
    Q1 = answ[0,0]/mat[0,0]
    size_mat = mat[:,0].size
    i = 1
    P = [P1]
    Q = [Q1]
    while i < size_mat-1:
        P.append(-mat[i,i+1]/(mat[i,i]+mat[i,i-1]*P[i-1]))
        Q.append((answ[i,0]-mat[i,i-1]*Q[i-1])/(mat[i,i]+mat[i,i-1]*P[i-1]))
        i+=1
    P.append(0)
    Q.append((answ[i,0]-mat[i,i-1]*Q[i-1])/(mat[i,i]+mat[i,i-1]*P[i-1]))
    return P, Q
```

```
def method(size_mat, P, Q):
    X = np.zeros(size_mat)
    X[size_mat-1] = Q[size_mat-1]
    i = size_mat-2
    while i > -1:
        X[i] = X[i+1]*P[i]+Q[i]
        i-=1
    return X

def run_method(mat,answ):
    size_mat = mat[:,0].size
    answ = np.asarray(answ).reshape(-1,1)
    P, Q = get_PQ(mat,answ)
    X = method(size_mat, P, Q)
    return X

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

def getRow(i,x,f):
    n = len(x)
    A = np.zeros([n-2])

    if i == 0:
        A[0] = 2*(h(1, x)+h(2, x))
        A[1] = h(2, x)
        B = 3*(((f[2] - f[1]) / h(2, x)) - ((f[1] - f[0]) / h(1, x)))
    elif i == n-3:
        A[i-1] = h(n-2, x)
        A[i] = 2*(h(n-2, x)+h(n-1, x))
        B = 3*(((y[n-1] - y[n-2]) / h(n-1, x)) - ((y[n-2] - y[n-3]) / h(n-2,
x)))
    elif i > 0:
        A[i-1] = h(i+1,x)
        A[i] = 2*(h(i+1, x)+h(i+2, x))
        A[i+1] = h(i+2,x)
        B = 3*(((y[i+2] - y[i+1]) / h(i+2, x)) - ((y[i+1] - y[i]) / h(i+1, x)))
    return A,B

def get_c(x, f):
    A = []
    B = []
    c = np.zeros(len(x)-1)
    for i in range(len(x)-2):
        a_tmp, b_tmp = getRow(i,x,f)
        A.append(a_tmp)
        B.append(b_tmp)
    c[1:] = run_method(np.array(A),np.array(B))
    return c
```

```
def get_a(x, y, c):
    n = len(x)-1
    a = np.zeros(n+1)
    for i in range(1, n):
        a[i-1] =  y[i-1]
    a[n-1] = y[n-1]
    return a

def get_b(x, y, c):
    n = len(x)-1
    b = np.zeros(n+1)
    for i in range(1, n):
        b[i-1] = ((y[i] - y[i-1])/h(i,x)) - ((h(i,x)*(c[i] + 2*c[i-1]))/3)
    b[n-1] = ( ((y[n] - y[n-1]) / h(n-1,x)) - 2*h(n-1,x)*c[n-1]/3 )
    return b

def get_d(x, y, c):
    n = len(x)-1
    d = np.zeros(n+1)
    for i in range(1, n):
        d[i-1] = (c[i] - c[i-1]) / (3*h(i,x))
    d[n-1] = (-c[n-1]/(3*h(n-1, x)))
    return d

def get_spline(X, a, b, c, d, x):
    n = len(x)
    j = 0
    for k in range(1,n):
        if ((x[k]>=X)and(x[k-1]<=X)):
            break;
        j+=1
    for i in range(n-1):
        spline = "{} + {}(x - {}) + {}(x - {})^2 + {}(x - {})^3".format(a[i],
b[i], x[i], c[i], x[i], d[i], x[i])
        if (i==j):
            spline+=" V"
        print(spline)
    t = X-x[j];
    return a[j] + b[j]*t + c[j]*t*t + d[j]*t*t*t

x = [0.1,0.5,0.9,1.3,1.7]
y = [-2.2026,-0.19315,0.79464,1.5624,2.2306]
X = 0.8
c = get_c(x,y)
a = get_a(x,y,c)
b = get_b(x,y,c)
d = get_d(x,y,c)
get_spline(X, a, b, c, d, x)
```

Задание 3

1) Постановка задачи:

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.

Вариант 16:

16.

i	0	1	2	3	4	5
x_i	0.1	0.5	0.9	1.3	1.7	2.1
y_i	-2.2026	-0.19315	0.79464	1.5624	2.2306	2.8419

2) Теория:

Дана таблично заданная функция, необходимо найти коэффициенты многочлена степени n, который максимально точно аппроксимирует данную функцию. Для этого необходимо добиться минимума функции

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

Для этого нужно решить систему из равенств нулю производных:

$$\frac{\partial \Phi}{\partial a_k} = 2 \sum_{j=0}^{N} \left[\sum_{i=0}^{n} a_i x_j^i - y_j \right] x_j^k = 0, \quad k = 0,1,\ldots,n .$$

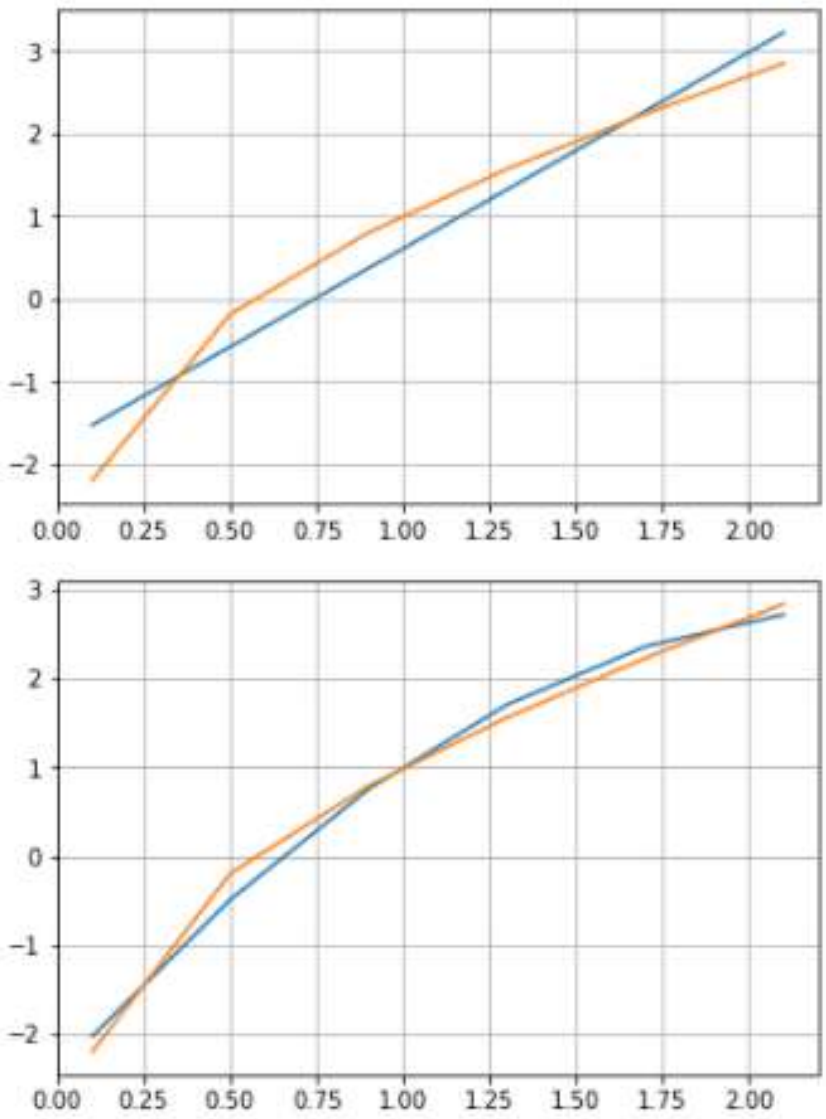
Полученное решение будет являться набором коэффициентов для искомого многочлена

3) Полученный ответ:

```
x =  [0.1, 0.5, 0.9, 1.3, 1.7, 2.1]
y =  [-2.2026, -0.19315, 0.79464, 1.5624, 2.2306, 2.8419]
Первой степени
(6.0a0)+(6.6a1)=5.03379
(6.6a0)+(10.06a1)=12.189471000000001
Значение функции в y: [-1.5368571428571425, -0.5865282857142855,
0.3638005714285715, 1.3141294285714284, 2.2644582857142854, 3.2147871428571424]
F1(x) = -1.7744393571428567 + (2.3758221428571424x^1)
(F-y)^2 = 0.9854121221771428

Второй степени
(6.0a0)+(6.6a1)+(10.06a2)=5.03379
```

```
(6.6a0)+(10.06a1)+(17.226a2)=12.189471000000001
(10.06a0)+(17.226a1)+(31.375000000000004a2)=22.193013899999997
Значение функции в y:  [-2.029066071428557, -0.4880865000000064,
0.7575677142857058, 1.7078965714285623, 2.362900071428567, 2.722578214285723]
F2(x) = -2.4604555513392667 + (4.406183973214236x^1) + (-0.9228917410714071x^2)
(F-y)^2 = 0.17138616750928573
```



4) Код программы:

```
import numpy as np
import sys
import lu
import matplotlib.pyplot as plt
```

```
def get_system(x,y,N):
    x = np.array(x)
    y = np.array(y)
    N += 1
    n = x.size
    A = np.zeros((N,N))
    B = np.zeros((N,1))
    for i in range(N):
        B[i] = np.sum(y*x**i)
        A[i,:] = [np.sum(x**(i+j)) for j in range(N)]
        if i == 0:
            A[0,0] = n
    return A,B


def print_system(A,B):
    A = A.astype(str)
    B = B.astype(str)
    A1 = []
    n = A[0].size
    for i in range(n):
        B[i] = str(B[i,0])
        A[i,:] = ["("+A[i,j]+"a"+str(j)+")" for j in range(n)]
        print("+".join(A[i,:])+"="+B[i,0])


def print_equation(x):
    n = len(x)
    res = "F"+str(n-1)+str("(x)")+" = "
    for i in range(n):
        if i == 0:
             res+=str(x[0])+" "
        else:
            res+="+ ("+str(x[i])+"x^"+str(i)+") "
    print(res)


def get_value(x,a):
    n = len(a)
    res = 0
    for i in range(n):
        res += x**i*a[i]
    return res

def get_deviation(x,y,a):
    n = len(x)
    res = 0
    for i in range(n):
        res += (get_value(x[i],a)-y[i])**2
    return res
```

```
def plot(x,y):
    fig, ax = plt.subplots()
    ax.plot(x,y)


def main():
    x = [0.1, 0.5, 0.9, 1.3, 1.7, 2.1]
    y = [-2.2026, -0.19315, 0.79464, 1.5624, 2.2306, 2.8419]
    n = len(x)
    print("x = ", x)
    print("y = ", y)
    print("Первой степени")
    A,B = get_system(x,y,1)
    print_system(A,B)
    a = lu.solve(A,B)
    y_new = [get_value(x[i],a) for i in range(n)]
    fig, ax = plt.subplots()
    ax.grid()
    ax.plot(x,y_new)
    ax.plot(x,y)
    print("Значение функции в y:", y_new)
    print_equation(a)
    print("(F-y)^2 = " + str(get_deviation(x,y,a))+"\n")
    print("Второй степени")
    A,B = get_system(x,y,2)
    print_system(A,B)
    a = lu.solve(A,B)
    y_new = [get_value(x[i],a) for i in range(n)]
    fig, ax = plt.subplots()
    ax.grid()
    ax.plot(x,y_new)
    ax.plot(x,y)
    print("Значение функции в y: ", y_new)
    print_equation(a)
    print("(F-y)^2 = " + str(get_deviation(x,y,a))+"\n")



if __name__ == "__main__":
    main()
```

Задание 4

1) Постановка задачи:

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

Вариант 16:

16. $X^* = 2.0$

i	0	1	2	3	4
x_i	0.0	1.0	2.0	3.0	4.0
y_i	0.0	2.0	3.4142	4.7321	6.0

2) Теория:

При аппроксимации функции многочленом первой степени первая производная вычисляется по формуле:

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

При аппроксимации многочленом второй степени:

$$y'(x) \approx \varphi'(x) = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} + \frac{\frac{y_{i+2} - y_{i+1}}{x_{i+2} - x_{i+1}} - \frac{y_{i+1} - y_i}{x_{i+1} - x_i}}{x_{i+2} - x_i}(2x - x_i - x_{i+1})$$

Формула для вычисления второй производной:

$$y''(x) \approx \varphi''(x) = 2\frac{\frac{y_{i+2} - y_{i+1}}{x_{i+2} - x_{i+1}} - \frac{y_{i+1} - y_i}{x_{i+1} - x_i}}{x_{i+2} - x_i}$$

3) Полученный ответ:

```
y' (left) = 1.4142000000000001
y' (right) = 1.3178999999999998
y'' = -0.09630000000000027
```

4) Код программы:

```
import numpy as np
def dx1_left(x, y, i):
    return (y[i+1] - y[i]) / (x[i+1] - x[i])

def dx1_right(x,y,i):
    return (y[i+2] - y[i+1]) / (x[i+2] - x[i+1])

def dx2(x, y, i):
    a = dx1_right(x,y,i)
    b = dx1_left(x,y,i)
    c = x[i+2] - x[i]
    return 2 * (a - b) /c

def main():
    x = [0.0, 1.0, 2.0, 3.0, 4.0]
    y = [0.0, 2.0, 3.4142, 4.7321, 6.0]
    X = 2.0
    for i in range(len(x)):
        if ((x[i]<=X) and (X <=x[i+1])):
            print("y' (left) = {}".format(dx1_left(x, y, i)))
            print("y' (right) = {}".format(dx1_right(x, y, i)))
            print("y'' = {}".format(dx2(x, y, i)))
            break

if __name__ == "__main__":
    main()
```

Задание 5

1) Постановка задачи:

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.

Вариант 16:

16. $y = \frac{x^2}{x^4 + 256}$, $X_0 = 0,\ X_k = 2,\ h_1 = 0.5,\ h_2 = 0.25;$

2) Теория:

Заменим подынтегральную функцию интерполяционным многочленом Лагранжа нулевой степени, проходящим через середину отрезка и получим формулу прямоугольников:

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{N} h_i\, f\left(\frac{x_{i-1}+x_i}{2}\right)$$

При замене функции многочленом Лагранжа первой степени, проходящим через концы отрезка интегрирования, получим формулу трапеций:

$$F = \int_a^b f(x)\,dx \approx \frac{1}{2}\sum_{i=1}^{N}(f_i + f_{i-1})\,h_i$$

При замене функции многочленом второй степени, проходящим через концы и середину отрезка, получим формулу Симпсона:

$$F = \int_a^b f(x)\,dx \approx \frac{1}{3}\sum_{i=1}^{N}(f_{i-1} + 4f_{i-\frac{1}{2}} + f_i)\,h_i$$

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

$$F = \int_a^b f(x)\,dx = F_h + \frac{F_h - F_{kh}}{k^p - 1} + O(h^{p+1})$$

3) Полученный ответ:

```
h = 0.5
Метод прямоугольника: 0.010012325651611462
Метод трапеции:0.010419470853869349
Метод Симпсона0.010144630551783072

h = 0.25
Метод прямоугольника: 0.01011442236874015
Метод трапеции:0.010215898252740406
Метод Симпсона0.010148040719030757

h = [0.5, 0.25]
Уточнение для метода прямоугольника: 0.010148454607783046
Уточнение для метода трапеции: 0.010148040719030759
Уточнение для метода Симпсона: 0.010148268063513936

Погрешность
Метод прямоугольника: 3.40322390428964e-05
Метод трапеции:6.785753370964695e-05
Метод Симпсона: 2.2734448317830724e-07
```

4) Код программы:

```
import numpy as np

def func(x):
    return x**2/(x**4+256)


def rectangle(f,x,h,n):
    res = 0
    for i in range(1,n):
        res += f((x[i]+x[i-1])/2)
    return h*res

def trapezoid(f,x,h,n):
    res = (f(x[0]) + f(x[n-1])) / 2;
    for i in range(1,n-1):
        res += f(x[i])
    return h*res

def sympson(f,x,h,n):
    res = f(x[0]) + f(x[n-1]);
    for i in range(1,n-1):
        if(i%2==0):
            res += 2*f(x[i])
        else:
            res += 4*f(x[i])
    return h*res/3

def rumb_romb_rich(Fh, Fkh, p):
    return Fh + ((Fh - Fkh) / (pow(2,p) - 1))


def get_x_n(X1,X0,h):
    N = int((X1-X0)/h+1)
    x = np.zeros(N)
    x[0] = X0
    for j in range(1,N):
            x[j] = x[j-1] + h
    return x, N

def main():
    x1 = 2
    x0 = 0
    h = [0.5, 0.25]
    n = len(h)
    r = np.zeros(n)
    t = np.zeros(n)
    s = np.zeros(n)
    rrr = np.zeros((n,3))
```

```
    for i in range(n):
        x, N = get_x_n(x1,x0,h[i])
        r[i] = rectangle(func,x, h[i], N)
        t[i] = trapezoid(func,x, h[i], N)
        s[i] = sympson(func, x, h[i], N)
        print("h = {}".format(h[i]))
        print("Метод прямоугольника: {}\nМетод трапеции:{}\nМетод
Симпсона{}\n".format(r[i],t[i],s[i]))
    for i in range(1,n):
        rrr[i-1][0] = rumb_romb_rich(r[i], r[i-1], 2)
        rrr[i-1][1] = rumb_romb_rich(t[i], t[i-1], 2)
        rrr[i-1][2] = rumb_romb_rich(s[i], s[i-1], 4)
        print("h = {}\nУточнение для метода прямоугольника: {}\nУточнение для
метода трапеции: {}\nУточнение для метода Симпсона: {}\n".format([h[i-1],h[i]],
rrr[i-1][0], rrr[i-1][1], rrr[i-1][2]))
        print("Погрешность\nМетод прямоугольника: {}\nМетод трапеции:{}\nМетод
Симпсона: {}\n".format(abs(r[i] - rrr[i-1][0]), abs(t[i] - rrr[i-1][1]),
abs(s[i] - rrr[i-1][2])))


if __name__ == "__main__":
    main()
```