МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное образовательное
учреждение высшего образования

«Национальный исследовательский университет «МЭИ»

Институт	ИВТИ
Кафедра	УИТ

ТИПОВОЙ РАСЧЕТ 1.

Дисциплина: Вычислительные Методы

Вариант 16

Студент гр. А-02-22		Синявский С.Ю
	(подпись)	
Преподаватель		Пепа Р.Ю.

Москва

2024

Задание 1

16. $Z = \dfrac{1}{\sqrt{4,00}} - 0,11^2 - 3,6$

$$Z(x_1, x_2, x_3) = \frac{1}{x_1} - x_2^2 - x_3$$

$$\begin{cases} x_1^* = \sqrt{4,00} & \Delta x_1^* \le 0,01 \\ x_2^* = 0,11^2 & \Delta x_2^* \le 2 \cdot 0,11 \cdot 0,005 \\ x_3^* = 3,6 & \Delta x_3^* \le 0,05 \end{cases}$$

$$\Delta Z^* = |f'_{x_1}| \cdot \Delta x_1^* + |f'_{x_2}| \Delta x_2^* + |f'_{x_3}| \Delta x_3^*$$

абс. погр. $\Delta Z^* \le \left| -\dfrac{1}{(\sqrt{4,00})^2} \right| \cdot 0,01 + |-2 \cdot 0,11| \cdot 2 \cdot 0,11 \cdot 0,005 + |1| \cdot 0,05 = 0,052742$

относ погр. $\delta_{Z^*} = \dfrac{\Delta Z^*}{Z} = \dfrac{0,052742}{+3,1121} = 0,0169474$

$Z = -\underbrace{3,1}_{\text{верные}}121 \pm 0,052742$

Задание 2.16

$f(x) = \dfrac{1}{x-2} - \sqrt{x} + 1$

x	0	1		
f(x)	0,5	-1		

Отрезок локализации $[a_0, b_0] = [0, 1]$

1 $x_0 = \dfrac{a_0 + b_0}{2} = \dfrac{1}{2}$

$\cancel{f\left(\tfrac{1}{2}\right) = -0,374 < 0}$ $f(a_0) = \dfrac{1}{2} > 0$

$\cancel{a_1 = \tfrac{1}{2},\ b_1 = 1}$ $f(x_0) = -0,37 < 0$

$\cancel{\dfrac{a_1 + b_1}{2} = 0,75}$

a	b	f(x_i)
0	0,5	-0,37
0	0,25	-0,071
0,125	0,25	0,113
0,1875	0,25	0,015
0,1875	0,21875	-0,0291
0,1875	0,203125	-0,00722

$\to b - a = 0,01562 < 2\varepsilon$

Ответ: $0,6094 \pm 0,01$

$x_6 = 0,6094$

Задача 3

$f = \frac{1}{x-2} - \sqrt{x} + 1$, $\varepsilon = 0{,}0001$

$\varphi(x) = x - \alpha f(x)$, $\alpha = \frac{2}{M+m}$

отрезок : $[0{,}1875 ; 0{,}25]$

$f'(x) = -\frac{1}{(x-2)^2} - \frac{1}{2\sqrt{x}}$; $m = f'(0{,}1875) = -1{,}4591$

$M = f'(\cancel{0{,}1875}\ 0{,}25) = -1{,}3265$

$\alpha = -0{,}7179$

$q = -0{,}0476$

$x_0 = 0{,}21875$

$x_1 = \varphi(x_0) = 0{,}21875 + 0{,}7179\left(-\frac{1}{(0{,}21875-2)^2} - \frac{1}{2\sqrt{0{,}21875}}\right) = -0{,}122489$

$+0{,}3412 \leq \frac{1-q}{q}\varepsilon = +0{,}0022$

$x_2 = \varphi(x_1) = 0{,}743771$

$x_3 = 0{,}705043$

$x_4 = 0{,}704424$

$|x_4 - x_3| \leq +0{,}0022$

Ответ: $0{,}704424 \pm 0{,}0001$

Задача 4

$f = x - 2e^{-x} - 1$ $a = 1, b = 3$, $e = 10^{-8}$

$x_0 = 2$

$x_1 = 2 - \frac{2 - 2e^{-2} - 1}{1 + 2e^{-2}} = 1{,}42603$

$x_2 = 1{,}46284$

$x_3 = 1{,}463055506$ $x_4 - x_3 = 7 \cdot 10^{-9} \leq 10^{-8}$

$x_4 = 1{,}463055513$ Ответ: $1{,}463055513 \pm 0{,}00000001$

Задача 5

$$\begin{pmatrix} -4 & 7 & -10 & -2 \\ -8 & 8 & -22 & 1 \\ -32 & 44 & -82 & -6 \\ -32 & 116 & -64 & -74 \end{pmatrix} \quad \begin{pmatrix} 23 \\ 47 \\ 186 \\ 182 \end{pmatrix}$$

$$\mu_{2,1} = \frac{a_{2,1}}{a_{1,1}} = \frac{-8}{-4} = 2 \qquad \mu_{3,1} = \frac{a_{3,1}}{a_{1,1}} = \frac{-32}{-4} = 8 \qquad \mu_{4,1} = \frac{a_{4,1}}{a_{1,1}} = 8$$

$$\begin{aligned} -4x_1 + 7x_2 - 10x_3 - 2x_4 &= 23 \\ 16x_2 - 2x_3 - 4x_4 &= 1 \\ -12x_2 - 2x_3 + 10x_4 &= 2 \\ 60x_2 + 16x_3 - 58x_4 &= -2 \end{aligned} \Rightarrow \dots \Rightarrow \begin{aligned} -4x_1 + 7x_2 - 10x_3 - 2x_4 &= 23 \\ 0 - 6x_2 - 2x_3 + 5x_4 &= 1 \\ 0 + 0 + 2x_3 + 0 &= 0 \\ 0 + 0 + 0 - 8x_4 &= 8 \end{aligned}$$

$$\begin{cases} x_4 = -1 \\ x_3 = 0 \\ x_2 = -1 \\ x_1 = -7 \end{cases}$$

Ответ: $\begin{pmatrix} -7 \\ -1 \\ 0 \\ -1 \end{pmatrix}$

Задача 8.16

$$\begin{cases} 8x_1 - 5x_2 = -73 \\ -6x_1 + 20x_2 - 5x_3 = 106 \\ -6x_2 + 22x_3 + 5x_4 = 72 \\ -2x_3 + 10x_4 + 4x_5 = -76 \\ -5x_4 + 10x_5 = 20 \end{cases}$$

$\alpha_1 = \frac{5}{8}$, $\beta_1 = -9\frac{1}{8}$, $\gamma_2 = \frac{65}{4}$

$\alpha_2 = \frac{4}{13}$, $\beta_2 = \frac{643}{65}$, $\gamma_3 = \frac{262}{13}$

$\alpha_3 = -\frac{65}{262}$, $\beta_3 = \frac{4269}{655}$, $\gamma_4 = \frac{1245}{131}$

$\alpha_4 = -\frac{524}{1245}$, $\beta_4 = -\frac{41242}{6225}$, $\beta_5 = -\frac{8171}{7535}$

$$\begin{cases} x_5 = \beta_5 = -1{,}084406 \\ x_4 = -6{,}168812 \\ x_3 = 8{,}047987 \\ x_2 = 12{,}368611 \\ x_1 = -1{,}394618 \end{cases}$$

Ответ: $\begin{pmatrix} -1,.. \\ 6,.. \\ 8,.. \\ 12,.. \\ -1,.. \end{pmatrix}$

Задание 9.16

$$\begin{pmatrix} 2,847 & -0,447 & 0 \\ 0,302 & -1,036 & 1,63 \\ 1,311 & 2,661 & -2,226 \end{pmatrix} \begin{pmatrix} -1,767 \\ 1,206 \\ -2,9 \end{pmatrix}$$

для матрицы A:

$$\|A\|_1 = \max_{1\le j\le n} \sum_{i=1} |a_{i,j}| = 4,46$$

$$\|A\|_\infty = \max_{1\le i\le m} \sum_{j=1}^{n} |a_{i,j}| = 6,198$$

$$\|A\|_2 = \sqrt{\lambda_{max}(A^T \cdot A)} = 4,115$$

$$\|A\|_F = 5,087 = \sqrt{\sum_{i=1}^{n} |A_{i,j}|^2} = 5,087$$

для вектора в:

абс. погр.: $5\cdot 10^{-3}$, $5\cdot 10^{-4}$, $5\cdot 10^{-2}$ соотв.

1) в норме $\|\cdot\|_1$: $\Delta(b) = 5\cdot 10^{-3} + 5\cdot 10^{-4} + 5\cdot 10^{-2} = 555\cdot 10^{-4} \approx 5,6\cdot 10^{-2}$

2) в норме $\|\cdot\|_2$: $\Delta(b) = 0,05$

3) в норме $\|\cdot\|_\infty$: $\Delta(b) = 5\cdot 10^{-2}$

$\|b_1\|_1 = 5,776$

$\|b\|_2 = 3,557$

$\|b\|_\infty = 2,9$

тогда

$$\delta_1 b = \frac{5,6\cdot 10^{-2}}{5,776} = 9,7\cdot 10^{-3}, \quad \delta_2 b = \frac{0,05}{3,557} = 0,014, \quad \delta_3 b = \frac{5\cdot 10^{-2}}{2,9} \approx 0,017$$

Задача 11.16

$$\begin{cases} -5x_1+2x_2-6x_3+104x_4=644 \\ 136x_1-7x_2+9x_3-3x_4=-16 \\ 7x_1+7x_2+155x_3-8x_4=-1034 \\ -5x_1+164x_2+9x_3-9x_4=-1420 \end{cases}$$

$$\begin{cases} 136x_1-7x_2+9x_3-3x_4=-16 \\ -5x_1+164x_2+9x_3-9x_4=-1420 \\ 7x_1+7x_2+155x_3-8x_4=-1034 \\ -5x_1+2x_2-6x_3+104x_4=644 \end{cases}$$

~~$x_1 = +\frac{7}{136}x_2 + \frac{9}{136}x_3 + \frac{3}{136}x_4 - \frac{16}{136}$~~

~~$x_2 = \frac{5}{164}x_2 + \frac{9}{5}x$~~

$$\begin{cases} x_1 = -\frac{16}{136} + \frac{7}{136}x_2 - \frac{9}{136}x_3 + \frac{3}{136}x_4 \\ x_2 = -\frac{1420}{164} + \frac{5}{164}x_1 - \frac{9}{164}x_3 + \frac{9}{164}x_4 \\ x_3 = -\frac{1034}{155} - \frac{7}{155}x_1 - \frac{7}{155}x_2 + \frac{8}{155}x_4 \\ x_4 = \frac{644}{104} + \frac{5}{104}x_1 - \frac{2}{104}x_2 + \frac{6}{104}x_3 \end{cases} \rightarrow B = \begin{pmatrix} 0 & \frac{7}{136} & -\frac{9}{136} & \frac{3}{136} \\ \frac{5}{164} & 0 & -\frac{9}{164} & \frac{9}{164} \\ -\frac{7}{155} & -\frac{7}{155} & 0 & \frac{8}{155} \\ \frac{5}{104} & -\frac{2}{104} & \frac{6}{104} & 0 \end{pmatrix} \begin{matrix} 0{,}13971 \\ 0{,}1402 \\ 0{,}1420 \\ 0{,}125 \end{matrix}$$

$$\|B\|_\infty = 0{,}1420$$

Поскольку норма итерац. матрицы B оказалась меньше 1, метод будет сходиться.

$$\|x^{(k+1)} - x^{(k)}\| \le \frac{1-\|B\|}{\|B\|}\varepsilon = \cancel{0{,}06}\ 0{,}17$$

$$x^{(0)} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

подставляя $x^{(0)}$ в правую часть системы, получим:

$$\begin{cases} x_1^{(1)} = -\frac{15}{136} \\ x_2^{(1)} = -\frac{1415}{164} \\ x_3^{(1)} = -\frac{208}{31} \\ x_4^{(1)} = \frac{653}{104} \end{cases} \qquad x^{(1)} = \begin{pmatrix} -\frac{15}{136} \\ -\frac{1415}{164} \\ -\frac{208}{31} \\ \frac{653}{104} \end{pmatrix}$$

2ой

Подставим $x^{(1)}$, получим:

$$\begin{cases} x_1^{(2)} = 0{,}02079 \\ x_2^{(2)} = -7{,}94911 \\ x_3^{(2)} = -5{,}95226 \\ x_4^{(2)} = 5{,}96583 \end{cases} \rightarrow \begin{pmatrix} 0{,}02079 \\ -7{,}94911 \\ -5{,}95226 \\ 5{,}96583 \end{pmatrix}$$

3ий

Подставим $x^{(2)}$, получим:

$$\begin{cases} x_1^{(3)} = -0{,}001294 \\ x_2^{(3)} = -7{,}93646 \\ x_3^{(3)} = -6{,}0050 \\ x_4^{(3)} = 6{,}00278 \end{cases}$$

$$\| x^{(3)} - \bar{x} \| \le \frac{\|B\|}{1 - \|B\|} \cdot \| x^{(3)} - x^{(2)} \|_\infty$$

Метод Зейделя

$$\begin{cases} x_1^{(k+1)} = -\frac{16}{136} + \frac{7}{136} x_2^{(k)} - \frac{9}{136} x_3^{(k)} + \frac{3}{136} x_4^{(k)} \\ x_2^{(k+1)} = -\frac{1420}{164} + \frac{5}{164} x_1^{(k+1)} - \frac{9}{164} x_3^{(k)} + \frac{9}{164} x_4^{(k)} \\ x_3^{(k+1)} = -\frac{1034}{155} - \frac{7}{155} x_1^{(k+1)} - \frac{7}{155} x_2^{(k+1)} + \frac{8}{155} x_4^{(k)} \\ x_4^{k+1} = \frac{644}{104} + \frac{5}{104} x_1^{(k+1)} - \frac{2}{104} x_2^{(k+1)} + \frac{6}{104} x_3^{(k+1)} \end{cases}$$

возьмем $x^{(0)} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$

1ый шаг:

$$\begin{cases} x_1^{(1)} = -0{,}110294 \\ x_2^{(1)} = -8{,}6619 \\ x_3^{(1)} = -6{,}22319 \\ x_4^{(1)} = 5{,}99455 \end{cases}$$

2ой шаг:

$$\begin{cases} x_1^{(2)} = -0{,}0194187 \\ x_2^{(2)} = -7{,}98864 \\ x_3^{(2)} = -5{,}99992 \\ x_4^{(2)} = 5{,}99885 \end{cases}$$

3ий шаг:

$$\begin{cases} x_1^{(3)} = 0{,}00055 \\ x_2^{(3)} = -8{,}00005 \\ x_3^{(3)} = -6{,}00006 \\ x_4^{(3)} = 6{,}00002 \end{cases} \qquad \bar{x} = \begin{pmatrix} 0 \\ -8 \\ -6 \\ 6 \end{pmatrix}$$

для метода Якоби:

~~норма невязок~~ $= \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$

для 1го:

$r^{(0)} = \begin{pmatrix} 644 \\ -16 \\ -1034 \\ -1420 \end{pmatrix} - \begin{pmatrix} -5 & 2 & -6 & 104 \\ 136 & -7 & 9 & -3 \\ 7 & 7 & 155 & -8 \\ -5 & 164 & 9 & -9 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 549 \\ -151 \\ -1195 \\ -1579 \end{pmatrix}$

$r^{(3)} = \ldots = \begin{pmatrix} -0{,}45267 \\ 0{,}674104 \\ 0{,}361518 \\ -10{,}35701 \end{pmatrix}$

$\dfrac{\|r^{(0)}\|}{\|r^{(3)}\|} = 293{,}28$ аналог. для Зейделя: $\dfrac{\|r^{0}\|}{\|r^{3}\|} = 37520$

Апостериорная оценка:

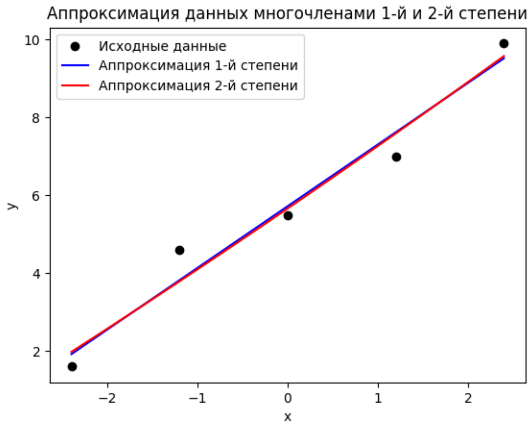
$\|x^{(3)} - \bar{x}\| \le \dfrac{\|B\|_\infty}{1 - \|B\|_\infty} \cdot \|x^3 - x^2\|_\infty$

$0{,}117614 \le \cancel{0{,}0174668}$ $0{,}02115208$ для Якоби

$0{,}00113 \le 0{,}00556$ для Зейделя

Ответы:
Задание 13

Формула для аппроксимации многочленом 1-й степени: y = 1.5833x + 5.7200
Формула для аппроксимации многочленом 2-й степени: y = 0.0198x^2 + 1.5833x + 5.6629
Среднеквадратичная ошибка для аппроксимации многочленом 1-й степени: 0.2576
Среднеквадратичная ошибка для аппроксимации многочленом 2-й степени: 0.2553



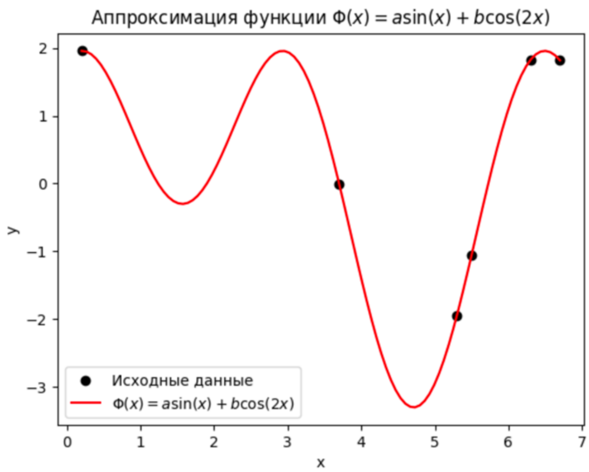
Задание 14

Коэффициенты аппроксимации: a = 1.4997, b = 1.8000

Среднеквадратичная ошибка: 0.0000

Исходные значения y: [1.956 -0.005 -1.942 -1.05   1.824 1.817]

Аппроксимированные значения y_approx: [1.95587279 -0.00518845 -1.9417527 -1.05011804 1.82422232 1.81710925]



Задание 15

Интерполяционный многочлен Лагранжа: -3*x**3/2 + 17*x**2 - 121*x/2 + 66
Интерполяционный многочлен Ньютона: -1.5*x**3 + 17.0*x**2 - 60.5*x + 66.0
Приближенное значение функции в точке 3.88 по методу Лагранжа: -0.431807999999975

Приближенное значение функции в точке 3.88 по методу Ньютона: -0.431807999999961

Задание 16

Значение интерполяционного многочлена первой степени в точке x~ = 2.57: 9.235
Значение интерполяционного многочлена второй степени в точке x~ = 2.57: 9.19834375
Значение интерполяционного многочлена третьей степени в точке x~ = 2.57: 9.1968927734375
Погрешность для многочлена первой степени: 0.03810722656249865
Погрешность для многочлена второй степени: 0.001450976562498596
Погрешность для многочлена третьей степени: 0.0

Задание 20

Метод центральных прямоугольников с h = 0.4: 1.173691
Метод трапеций с h = 0.4: 1.174005
Метод трапеций с h = 0.2: 1.370428
Оценка погрешности по правилу Рунге для метода трапеций: 0.065475
Уточненный результат для метода трапеций: 1.435903
Метод Симпсона с h = 0.4: 1.567274

Задание 21

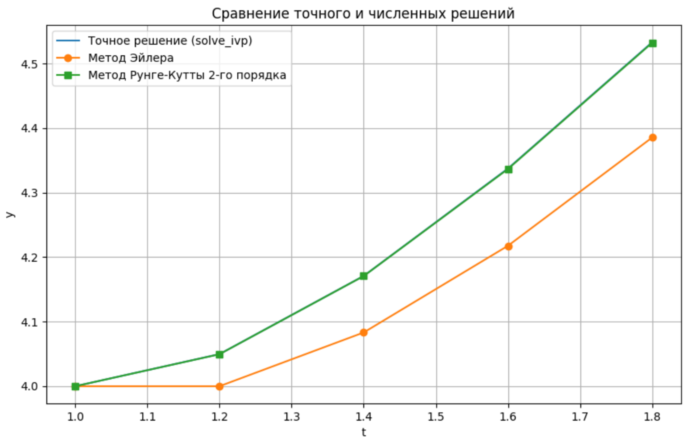
Точное значение интеграла: 1.60369166666667
Оптимальный шаг h для достижения точности ε = 0.01: 0.1
Приближенное значение интеграла методом трапеций с шагом h = 0.1: 1.599045
Погрешность приближенного значения: 0.004647

Задание 23

Центральная разностная производная в точке x0 = 0.15: 1.326000
Левая разностная производная в точке x0 = 0.15: 1.917500
Вторая разностная производная в точке x0 = 0.15: -11.830000
Точное значение первой производной в точке x0 = 0.15: 1.338000
Точное значение второй производной в точке x0 = 0.15: -11.890000
Погрешность центральной разностной производной: 0.012000
Погрешность левой разностной производной: 0.579500
Погрешность второй разностной производной: 0.060000

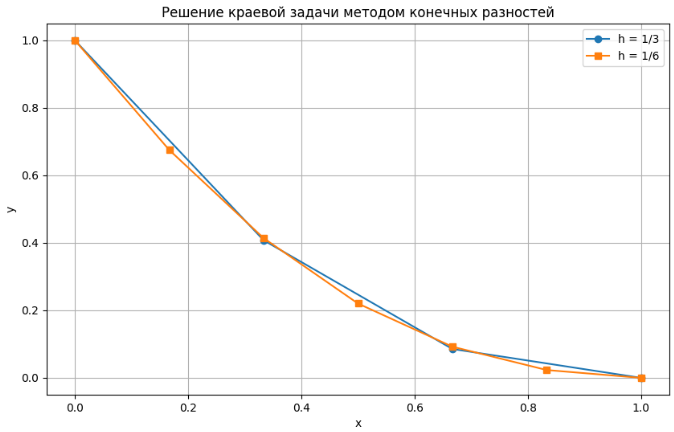
Задание 24

Погрешность метода Эйлера: [0. ; 0.02479339 ; 0.04371852 ; 0.05948783 ; 0.0734117]
Погрешность метода Рунге-Кутты 2-го порядка: [0. ; 0.00011999 ; 0.00019119 ; 0.00024324 ; 0.00028676]



Задание 27

Погрешность по правилу Рунге: [0. ; 0.00211229 ; 0.00228113 ; 0.]



 # 13 Задание

import numpy as np
import matplotlib.pyplot as plt

# Данные для 16 варианта
x = np.array([-2.4, -1.2, 0, 1.2, 2.4])
y = np.array([1.6, 4.6, 5.5, 7, 9.9])

# Аппроксимация многочленом первой степени
coeffs_1 = np.polyfit(x, y, 1)
poly_1 = np.poly1d(coeffs_1)
y_approx_1 = poly_1(x)

# Аппроксимация многочленом второй степени
coeffs_2 = np.polyfit(x, y, 2)
poly_2 = np.poly1d(coeffs_2)
y_approx_2 = poly_2(x)

Среднеквадратичная ошибка

```
mse_1 = np.mean((y - y_approx_1)**2)
mse_2 = np.mean((y - y_approx_2)**2)

# Вывод формул многочленов
print(f'Формула для аппроксимации многочленом 1-й степени: y = {coeffs_1[0]:.4f}x +
{coeffs_1[1]:.4f}')
print(f'Формула для аппроксимации многочленом 2-й степени: y = {coeffs_2[0]:.4f}x^2 +
{coeffs_2[1]:.4f}x + {coeffs_2[2]:.4f}')

# Вывод среднеквадратичных ошибок
print(f'Среднеквадратичная ошибка для аппроксимации многочленом 1-й степени:
{mse_1:.4f}')
print(f'Среднеквадратичная ошибка для аппроксимации многочленом 2-й степени:
{mse_2:.4f}')

# Построение графиков
plt.scatter(x, y, color='black', label='Исходные данные')
x_line = np.linspace(min(x), max(x), 100)
plt.plot(x_line, poly_1(x_line), label='Аппроксимация 1-й степени', color='blue')
plt.plot(x_line, poly_2(x_line), label='Аппроксимация 2-й степени', color='red')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.title('Аппроксимация данных многочленами 1-й и 2-й степени')
plt.show()
```

 # 14 Задание

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

# Данные для 16 варианта
x = np.array([0.2, 3.7, 5.3, 5.5, 6.3, 6.7])
y = np.array([1.956, -0.005, -1.942, -1.05, 1.824, 1.817])

# Построение матрицы для метода наименьших квадратов
A = np.vstack([np.sin(x), np.cos(2 * x)]).T
coeffs, residuals, _, _ = np.linalg.lstsq(A, y, rcond=None)

# Коэффициенты аппроксимации
a, b = coeffs

# Функция аппроксимации
def phi(x):
    return a * np.sin(x) + b * np.cos(2 * x)

# Вычисление значений аппроксимированной функции
y_approx = phi(x)

# Среднеквадратичная ошибка
mse = np.mean((y - y_approx) ** 2)
```

```
print(f'Коэффициенты аппроксимации: a = {a:.4f}, b = {b:.4f}')
print(f'Среднеквадратичная ошибка: {mse:.4f}')

# Проверка, что значения y_approx не совпадают идеально с y
print(f'Исходные значения y: {y}')
print(f'Аппроксимированные значения y_approx: {y_approx}')

# Построение графиков
plt.scatter(x, y, color='black', label='Исходные данные')
x_line = np.linspace(min(x), max(x), 100)
plt.plot(x_line, phi(x_line), label='$\Phi(x) = a \sin(x) + b \cos(2x)$', color='red')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.title('Аппроксимация функции $\Phi(x) = a \sin(x) + b \cos(2x)$')
plt.show()
```

 # Задание 15

```
import numpy as np
import sympy as sp

# Данные для 16 варианта
x = np.array([2, 3, 4, 5])
y = np.array([1, -3, 0, 1])
x_approx = 3.88

# Интерполяционный многочлен Лагранжа
def lagrange_polynomial(x, y):
    n = len(x)
    X = sp.symbols('x')
    L = 0
    for i in range(n):
        term = y[i]
        for j in range(n):
            if j != i:
                term *= (X - x[j]) / (x[i] - x[j])
        L += term
    return sp.simplify(L)

# Интерполяционный многочлен Ньютона
def newton_polynomial(x, y):
    def divided_differences(x, y):
        n = len(y)
        coef = np.zeros([n, n])
        coef[:,0] = y
        for j in range(1,n):
            for i in range(n-j):
                coef[i][j] = (coef[i+1][j-1] - coef[i][j-1]) / (x[i+j] - x[i])
        return coef[0, :]
```

```
    coef = divided_differences(x, y)
    X = sp.symbols('x')
    n = len(coef) - 1
    P = coef[0]
    for k in range(1, n + 1):
        term = coef[k]
        for j in range(k):
            term *= (X - x[j])
        P += term
    return sp.simplify(P)

# Вычисление многочленов
L_poly = lagrange_polynomial(x, y)
N_poly = newton_polynomial(x, y)

# Вывод многочленов
print(f'Интерполяционный многочлен Лагранжа: {L_poly}')
print(f'Интерполяционный многочлен Ньютона: {N_poly}')

# Вычисление значений
y_lagrange = L_poly.subs('x', x_approx)
y_newton = N_poly.subs('x', x_approx)

print(f'Приближенное значение функции в точке {x_approx} по методу Лагранжа: {y_lagrange}')
print(f'Приближенное значение функции в точке {x_approx} по методу Ньютона: {y_newton}')
```

 # Задание 16

```
import numpy as np

# Данные
x = np.array([2.4, 2.8, 2, 3.6, 4])
y = np.array([8.3, 10.5, 6.4, 15.9, 19])
x_tilde = 2.57

# Сортировка данных по расстоянию до x_tilde
distances = np.abs(x - x_tilde)
sorted_indices = np.argsort(distances)
x = x[sorted_indices]
y = y[sorted_indices]


# Вычисление разделенных разностей для многочлена Ньютона
def divided_differences(x, y):
    n = len(y)
    coef = np.zeros([n, n])
    coef[:, 0] = y

    for j in range(1, n):
```

```
        for i in range(n - j):
            coef[i][j] = (coef[i + 1][j - 1] - coef[i][j - 1]) / (x[i + j] - x[i])

    return coef[0, :]


# Интерполяционный многочлен Ньютона
def newton_polynomial(x, y, x_tilde):
    coef = divided_differences(x, y)
    n = len(x) - 1
    p = coef[n]

    for k in range(1, n + 1):
        p = coef[n - k] + (x_tilde - x[n - k]) * p

    return p


# Многочлены Ньютона первой, второй и третьей степени
P1 = newton_polynomial(x[:2], y[:2], x_tilde)
P2 = newton_polynomial(x[:3], y[:3], x_tilde)
P3 = newton_polynomial(x[:4], y[:4], x_tilde)

# Печать результатов
print(f"Значение интерполяционного многочлена первой степени в точке x~ = {x_tilde}: {P1}")
print(f"Значение интерполяционного многочлена второй степени в точке x~ = {x_tilde}: {P2}")
print(f"Значение интерполяционного многочлена третьей степени в точке x~ = {x_tilde}: {P3}")


# Оценка погрешности
def estimate_error(actual, interpolated):
    return abs(actual - interpolated)


# Предположим, что значение функции в x_tilde можно оценить с помощью интерполяции
# Для оценки погрешности используем значение P3 как более точное
error_P1 = estimate_error(P3, P1)
error_P2 = estimate_error(P3, P2)

# Если известное точное значение функции в x_tilde неизвестно, оценим погрешность для многочлена третьей степени
# как наибольшее отклонение от известных значений
actual_values = [8.3, 10.5, 6.4, 15.9, 19]
predicted_values = [newton_polynomial(x, y, xi) for xi in x]
error_P3 = max([estimate_error(av, pv) for av, pv in zip(actual_values, predicted_values)])

print(f"Погрешность для многочлена первой степени: {error_P1}")
print(f"Погрешность для многочлена второй степени: {error_P2}")
print(f"Погрешность для многочлена третьей степени: {error_P3}")
```

 # Задание 20

```
import numpy as np

# Функция для интегрирования
def f(x):
    return np.cos(1 / x)

# Параметры задачи
a = 4.2
b = 5.8

# Метод центральных прямоугольников
def central_rectangles(f, a, b, h):
    n = int((b - a) / h)
    result = 0.0
    for i in range(n):
        x_i = a + i * h
        x_i_star = x_i + h / 2
        result += f(x_i_star)
    result *= h
    return result

# Метод трапеций
def trapezoidal(f, a, b, h):
    n = int((b - a) / h)
    result = (f(a) + f(b)) / 2
    for i in range(1, n):
        result += f(a + i * h)
    result *= h
    return result

# Метод Симпсона
def simpson(f, a, b, h):
    n = int((b - a) / h)
    if n % 2 != 0:  # n должно быть четным для метода Симпсона
        n += 1
        h = (b - a) / n
    result = f(a) + f(b)
    for i in range(1, n):
        x_i = a + i * h
        if i % 2 == 0:
            result += 2 * f(x_i)
        else:
            result += 4 * f(x_i)
    result *= h / 3
    return result

# Вычисление интегралов
h1 = 0.4
```

```
h2 = 0.2

I_central_rectangles = central_rectangles(f, a, b, h1)
I_trapezoidal_h1 = trapezoidal(f, a, b, h1)
I_trapezoidal_h2 = trapezoidal(f, a, b, h2)
I_simpson = simpson(f, a, b, h1)

# Оценка погрешности по правилу Рунге для метода трапеций
def runge_error(I_h1, I_h2, k):
    return abs(I_h2 - I_h1) / (2**k - 1)

k = 2  # Порядок метода трапеций
error_trapezoidal = runge_error(I_trapezoidal_h1, I_trapezoidal_h2, k)

# Уточненный результат для метода трапеций
I_trapezoidal_refined = I_trapezoidal_h2 + (I_trapezoidal_h2 - I_trapezoidal_h1) / (2**k - 1)

# Печать результатов
print(f"Метод центральных прямоугольников с h = 0.4: {I_central_rectangles:.6f}")
print(f"Метод трапеций с h = 0.4: {I_trapezoidal_h1:.6f}")
print(f"Метод трапеций с h = 0.2: {I_trapezoidal_h2:.6f}")
print(f"Оценка погрешности по правилу Рунге для метода трапеций:
{error_trapezoidal:.6f}")
print(f"Уточненный результат для метода трапеций: {I_trapezoidal_refined:.6f}")
print(f"Метод Симпсона с h = 0.4: {I_simpson:.6f}")
```

 # Задание 21

```
import numpy as np
from sympy import symbols, diff, integrate

# Заданные параметры
a = -0.1
b = 0.4
c0 = 3
c1 = 3
c2 = -5
c3 = -3
c4 = 3

# Функция для интегрирования
def f(x):
    return c0 + c1*x + c2*x**2 + c3*x**3 + c4*x**4

# Точное значение интеграла с использованием SymPy
x = symbols('x')
exact_integral = integrate(c0 + c1*x + c2*x**2 + c3*x**3 + c4*x**4, (x, a, b))
exact_value = exact_integral.evalf()
print(f"Точное значение интеграла: {exact_value}")

# Метод трапеций
```

```
def trapezoidal(f, a, b, h):
    n = int((b - a) / h)
    result = (f(a) + f(b)) / 2
    for i in range(1, n):
        result += f(a + i * h)
    result *= h
    return result

# Оценка погрешности априорно по второй производной
x = symbols('x')
polynomial = c0 + c1*x + c2*x**2 + c3*x**3 + c4*x**4
second_derivative = diff(polynomial, x, x)
max_second_derivative = max(abs(second_derivative.subs(x, a)), abs(second_derivative.subs(x,
b)))

# Априорная оценка погрешности
def trapezoidal_error(b, a, h, max_second_derivative):
    return ((b - a) * h**2 / 12) * max_second_derivative

# Подбор шага h для достижения погрешности не более 0.01
epsilon = 0.01
h = 0.1  # Начальное значение шага
error = trapezoidal_error(b, a, h, max_second_derivative)

while error > epsilon:
    h /= 2
    error = trapezoidal_error(b, a, h, max_second_derivative)

# Вычисление интеграла с найденным шагом
I_trapezoidal = trapezoidal(f, a, b, h)

# Печать результатов
print(f"Оптимальный шаг h для достижения точности ε = {epsilon}: {h}")
print(f"Приближенное значение интеграла методом трапеций с шагом h = {h}:
{I_trapezoidal:.6f}")
print(f"Погрешность приближенного значения: {abs(exact_value - I_trapezoidal):.6f}")
```

 # Задание 23

```
import numpy as np
from sympy import symbols, diff

# Заданные параметры
a = -0.1
b = 0.4
h = 0.1
x0 = round((a + b) / 2, 2)

# Функция для интегрирования
def f(x):
    return 3 + 3*x - 5*x**2 - 3*x**3 + 3*x**4
```

```
# Центральная разностная производная
def central_difference(f, x, h):
    return (f(x + h) - f(x - h)) / (2 * h)

# Левая разностная производная
def left_difference(f, x, h):
    return (f(x) - f(x - h)) / h

# Вторая разностная производная
def second_difference(f, x, h):
    return (f(x + h) - 2 * f(x) + f(x - h)) / h**2

# Вычисление производных
f_prime_central = central_difference(f, x0, h)
f_prime_left = left_difference(f, x0, h)
f_double_prime = second_difference(f, x0, h)

# Вычисление точного значения производных с использованием SymPy
x = symbols('x')
polynomial = 3 + 3*x - 5*x**2 - 3*x**3 + 3*x**4
exact_f_prime = diff(polynomial, x)
exact_f_double_prime = diff(polynomial, x, x)
exact_f_prime_value = exact_f_prime.subs(x, x0).evalf()
exact_f_double_prime_value = exact_f_double_prime.subs(x, x0).evalf()

# Печать результатов
print(f"Центральная разностная производная в точке x0 = {x0}: {f_prime_central:.6f}")
print(f"Левая разностная производная в точке x0 = {x0}: {f_prime_left:.6f}")
print(f"Вторая разностная производная в точке x0 = {x0}: {f_double_prime:.6f}")
print(f"Точное значение первой производной в точке x0 = {x0}: {exact_f_prime_value:.6f}")
print(f"Точное значение второй производной в точке x0 = {x0}:
{exact_f_double_prime_value:.6f}")

# Сравнение точных и приближенных значений
error_central = abs(exact_f_prime_value - f_prime_central)
error_left = abs(exact_f_prime_value - f_prime_left)
error_second = abs(exact_f_double_prime_value - f_double_prime)

print(f"Погрешность центральной разностной производной: {error_central:.6f}")
print(f"Погрешность левой разностной производной: {error_left:.6f}")
print(f"Погрешность второй разностной производной: {error_second:.6f}")
```

 # Задание 24

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp

# Заданные параметры
t0 = 1
```

```
y0 = 4
h = 0.2
t_end = t0 + 0.8

# Определение функции
def f(t, y):
    return y / t - (t + 3) / t**2

# Метод Эйлера
def euler_method(f, t0, y0, h, t_end):
    t_values = np.arange(t0, t_end + h, h)
    y_values = np.zeros(len(t_values))
    y_values[0] = y0
    for i in range(1, len(t_values)):
        y_values[i] = y_values[i - 1] + h * f(t_values[i - 1], y_values[i - 1])
    return t_values, y_values

# Метод Рунге-Кутты 2-го порядка
def runge_kutta_2nd_order(f, t0, y0, h, t_end):
    t_values = np.arange(t0, t_end + h, h)
    y_values = np.zeros(len(t_values))
    y_values[0] = y0
    for i in range(1, len(t_values)):
        k1 = f(t_values[i - 1], y_values[i - 1])
        k2 = f(t_values[i - 1] + h / 2, y_values[i - 1] + h / 2 * k1)
        y_values[i] = y_values[i - 1] + h * k2
    return t_values, y_values

# Решение точное с использованием solve_ivp
sol = solve_ivp(f, [t0, t_end], [y0], method='RK45', t_eval=np.arange(t0, t_end + h, h))

# Решения методами Эйлера и Рунге-Кутты
t_euler, y_euler = euler_method(f, t0, y0, h, t_end)
t_rk2, y_rk2 = runge_kutta_2nd_order(f, t0, y0, h, t_end)

# Решения для шага h/2
_, y_euler_half = euler_method(f, t0, y0, h/2, t_end)
_, y_rk2_half = runge_kutta_2nd_order(f, t0, y0, h/2, t_end)

# Оценка погрешности по правилу Рунге
def runge_error(y_half, y_full, k):
    return np.abs(y_half[::2] - y_full) / (2**k - 1)

# Погрешности
error_euler = runge_error(y_euler_half, y_euler, 1)
error_rk2 = runge_error(y_rk2_half, y_rk2, 2)

# Графики решений
plt.figure(figsize=(10, 6))
plt.plot(sol.t, sol.y[0], label='Точное решение (solve_ivp)')
plt.plot(t_euler, y_euler, 'o-', label='Метод Эйлера')
plt.plot(t_rk2, y_rk2, 's-', label='Метод Рунге-Кутты 2-го порядка')
```

```
plt.xlabel('t')
plt.ylabel('y')
plt.legend()
plt.title('Сравнение точного и численных решений')
plt.grid(True)
plt.show()

# Печать результатов
print(f"Погрешность метода Эйлера: {error_euler}")
print(f"Погрешность метода Рунге-Кутты 2-го порядка: {error_rk2}")
```

 # Задание 27

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

# Заданные параметры
a = 0
b = 1
y_a = 1
y_b = 0


# Функции q(x) и f(x)
def q(x):
    return x


def f(x):
    return 2 + x - 2 * x ** 2


# Метод конечных разностей
def finite_difference_method(a, b, y_a, y_b, q, f, h):
    n = int((b - a) / h)
    x = np.linspace(a, b, n + 1)
    A = np.zeros((n + 1, n + 1))
    B = np.zeros(n + 1)

    A[0, 0] = 1
    A[n, n] = 1
    B[0] = y_a
    B[n] = y_b

    for i in range(1, n):
        A[i, i - 1] = 1 / h ** 2 - q(x[i]) / (2 * h)
        A[i, i] = -2 / h ** 2 + q(x[i])
        A[i, i + 1] = 1 / h ** 2 + q(x[i]) / (2 * h)
        B[i] = f(x[i])

    y = np.linalg.solve(A, B)
```

```python
    return x, y


# Вычисление решений
h1 = 1 / 3
h2 = 1 / 6

x1, y1 = finite_difference_method(a, b, y_a, y_b, q, f, h1)
x2, y2 = finite_difference_method(a, b, y_a, y_b, q, f, h2)


# Оценка погрешности по правилу Рунге
def runge_error(y1, y2, k):
    return np.abs(y1 - y2[::2]) / (2 ** k - 1)


error = runge_error(y1, y2, 2)

# Графики решений
plt.figure(figsize=(10, 6))
plt.plot(x1, y1, 'o-', label='h = 1/3')
plt.plot(x2, y2, 's-', label='h = 1/6')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.title('Решение краевой задачи методом конечных разностей')
plt.grid(True)
plt.show()

# Печать результатов
print(f"Погрешность по правилу Рунге: {error}")
```