Министерство образования и науки Российской Федерации

Федеральное агентство по образованию

Московский авиационный институт

(Государственный технический университет)

Кафедра 311

Выполнила:

Студентка гр. М3О-216Бк-22

Экзамен 15.06.24

Вариант 8

Хутиева Эрика Арсеновна

Численные методы

Москва 2024

https://github.com/erikahutieva/Python/tree/main/Chislaki все программы вы можете найти по данной ссылке

1. Определить корень уравнения методом половинного деления с точностью $\varepsilon = 0{,}001$

8	$\sin\left(x+\pi/3\right)-0{,}5x=0$

1task.py > ...

```python
from math import *

def y (x):
    return sin(x+pi/3)-0.5*x

a=float(input("Введите число a "))
b=float(input("Введите число b "))
e=float(input("Введите точность "))

while y(a)*y(b)>=0:
    print("Функция не меняет знак")
    a=float(input("Введите число a "))
    b=float(input("Введите число b "))
    e=float(input("Введите точность "))
count=0

while b-a>=e:
    count+=1
    c=(a+b)/2
    if y(c) == 0:
        print(f' c = {c},f(c) = {y(c)}')
    else:
        print(f'Итерация {count}: a = {a}, b = {b}, c = {c}, f(c) = {y(c)}')
        if y(c)*y(a)<0:
            b=c
        else:
            a=c
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
/usr/local/bin/python3.12 /Users/ricardof/Desktop/Числаки/1task.py
ricardof@ox-i1 Числаки % /usr/local/bin/python3.12 /Users/ricardof/Desktop/Числаки/1task.py
Введите число a 1
Введите число b 3
Введите точность 0.001
Итерация 1: a = 1.0, b = 3.0, c = 2.0, f(c) = -0.9057450187415146
Итерация 2: a = 1.0, b = 2.0, c = 1.5, f(c) = -0.1899922930611887
Итерация 3: a = 1.0, b = 1.5, c = 1.25, f(c) = 0.12256948589341898
Итерация 4: a = 1.25, b = 1.5, c = 1.375, f(c) = -0.028570214121922177
Итерация 5: a = 1.25, b = 1.375, c = 1.3125, f(c) = 0.04837540945815366
Итерация 6: a = 1.3125, b = 1.375, c = 1.34375, f(c) = 0.010235632396581651
Итерация 7: a = 1.34375, b = 1.375, c = 1.359375, f(c) = -0.009085431924101828
Итерация 8: a = 1.34375, b = 1.359375, c = 1.3515625, f(c) = 0.0005957415189332682
Итерация 9: a = 1.3515625, b = 1.359375, c = 1.35546875, f(c) = -0.004239706852580016
Итерация 10: a = 1.3515625, b = 1.35546875, c = 1.353515625, f(c) = -0.0018206953268450343
Итерация 11: a = 1.3515625, b = 1.353515625, c = 1.3525390625, f(c) = -0.0006121547254381854
ricardof@ox-i1 Числаки % /usr/local/bin/python3.12 /Users/ricardof/Desktop/Числаки/1task.py
Введите число a 1
Введите число b 1
Введите точность 0.01
Функция не меняет знак
Введите число a
```

2. Решить уравнение методом Ньютона и хорд с точностью $\varepsilon = 0{,}001$.

8	$x^3 + 3x - 1 = 0$

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Ньютон

$$x_{n+1} = x_n - f(x_n) \cdot \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}$$

хорд

2task.py > ...

```python
def newton(f, df, x0):
    e=0.001
    iter=100
    x_n = float(x0)
    for i in range(iter):
        f_x_n = f(x_n)
        df_x_n = df(x_n)
        x_next = x_n - f_x_n / df_x_n

        if abs(x_next - x_n) < e:
            return x_next

        x_n = x_next

    return "Решение не сходится."

def hord(f, x0, x1, e=0.001, iter=100):
    f_x0 = f(x0)
    f_x1 = f(x1)

    for n in range(iter):
        x_next = x1 - f_x1 * (x1 - x0) / (f_x1 - f_x0)

        if abs(x_next - x1) < e:
            return float(x_next)

        x0, x1 = x1, x_next
        f_x0, f_x1 = f_x1, f(x_next)

    return float(x1)


f = lambda x: x**3 + 3*x-1
df = lambda x: 3*x**2 + 3
print("Метод Ньютона:", newton(f, df, x0=1.0))
print("Метод хорд:", hord(f, x0=1.0, x1=2.0))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
/usr/local/bin/python3.12 /Users/ricardof/Desktop/Числаки/2task.py
ricardof@ox-i1 Числаки % /usr/local/bin/python3.12 /Users/ricardof/Desktop/Числаки/2task.py
Метод Ньютона: 0.32218535502284523
Метод хорд: 0.32218536774452516
ricardof@ox-i1 Числаки %
```

3. Решить систему $x = Cx + d$ методом простой итерации и Зейделя с точностью ε = 0,001

8	$\begin{pmatrix} 0 & 0{,}1 & -0{,}1 & 0{,}2 \\ 0{,}2 & 0 & -0{,}2 & 0{,}1 \\ 0{,}13 & -0{,}2 & 0 & 0{,}3 \\ 0{,}1 & -0{,}1 & -0{,}2 & 0 \end{pmatrix}$	$\begin{pmatrix} -1 \\ -1 \\ 2 \\ 0{,}1 \end{pmatrix}$

Метод простой итерации

1. **Приведение системы уравнений к итерационной форме**:
   Представим систему $A\mathbf{x} = \mathbf{b}$ в виде $\mathbf{x} = B\mathbf{x} + \mathbf{c}$.

   Для этого преобразуем систему к виду $\mathbf{x}^{(k+1)} = C\mathbf{x}^{(k)} + \mathbf{d}$, где матрица C и вектор $\mathbf{d}$ получаются из A и $\mathbf{b}$.

2. **Начальное приближение**:
   Выберем начальное приближение $\mathbf{x}^{(0)}$.

3. **Итерационный процесс**:

$$\mathbf{x}^{(k+1)} = C\mathbf{x}^{(k)} + \mathbf{d}$$

   Повторяем до тех пор, пока $\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\| < \varepsilon$.

Метод Зейделя

Метод Зейделя — это модификация метода простой итерации, в которой используются уже вычисленные значения на текущем шаге.

1. **Приведение системы уравнений к итерационной форме**:
   Как и в методе простой итерации, представим систему $A\mathbf{x} = \mathbf{b}$ в виде $\mathbf{x} = B\mathbf{x} + \mathbf{c}$.

2. **Начальное приближение**:
   Выберем начальное приближение $\mathbf{x}^{(0)}$.

3. **Итерационный процесс**:
   Для каждого уравнения обновляем значения по схеме:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right)$$

   Повторяем до тех пор, пока $\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\| < \varepsilon$.

3task.py > ...

```python
def preobraz(C,d):
    n = len(C)
    m = len(d)
    A = [[0 for _ in range(n)] for _ in range(n)]
    for i in range(n):
        for j in range(n):
            if i == j:
                A[i][j] = 1 - C[i][j]
            else:
                A[i][j] = -C[i][j]
    return A

def simple_iteration(C, d):
    e = 0.001
    n = len(C)
    x = [0] * n

    norm = e+1
    iteration_count = 0

    while norm >= e:
        x_new = [0] * n
        for i in range(n):
            sum_Cx = sum(C[i][j] * x[j] for j in range(n))
            x_new[i] = sum_Cx + d[i]
        norm = sum((x_new[i] - x[i]) ** 2 for i in range(n)) ** 0.5
        x = x_new
        iteration_count += 1
    return x

def zeidel(C, d):
    C=preobraz(C,d)
    e = 0.001
    n = len(C)
    x = [0] * n
    norm = e + 1
    while norm >= e:
        x_new = x.copy()
        for i in range(n):
            sum1 = sum(C[i][j] * x_new[j] for j in range(i))
            sum2 = sum(C[i][j] * x[j] for j in range(i + 1, n))
            x_new[i] = (d[i] - sum1 - sum2)

        norm = sum((x_new[i] - x[i]) ** 2 for i in range(n)) ** 0.5
        x = x_new
    return x

C = [
    [0, 0.1, -0.1, 0.2],
    [0.2, 0, -0.2, 0.1],
    [0.13, -0.2, 0, 0.3],
    [0.1, -0.1, -0.2, 0]
]
d = [-1, -1, 2, 0.1]

print("Решение:", simple_iteration(C, d))
print("Решение:", zeidel(C, d))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
/usr/local/bin/python3.12 /Users/ricardof/Desktop/Числаки/3task.py
ricardof@ox-i1 Числаки % /usr/local/bin/python3.12 /Users/ricardof/Desktop/Числаки/3task.py
Решение: [-1.437506985, -1.730723024, 2.0736266984, -0.285416794]
Решение: [-1.4375494607611536, -1.7307747655651635, 2.0736585264736687, -0.2854091748143327]
ricardof@ox-i1 Числаки %
```

4. Решить систему методом простой итерации и Ньютона с точностью $\varepsilon = 0{,}001$.

8	$\begin{cases} 2x - \cos(y+1) = 0 \\ y + \sin(x) = -0{,}4 \end{cases}$

Метод простой итерации:

4task.py > ...

```python
import math

def simple_iteration():
    e = 0.001
    x, y = 0, 0  # Начальные значения
    iteration_count = 0

    while True:
        x_new = math.cos(y - 1) / 2
        y_new = -0.4 - math.sin(x)

        if abs(x_new - x) < e and abs(y_new - y) < e:
            break

        x, y = x_new, y_new
        iteration_count += 1

    return x, y

x_solution, y_solution = simple_iteration()
print("Решение:")
print("x =", x_solution)
print("y =", y_solution)

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
/usr/local/bin/python3.12 /Users/ricardof/Desktop/Числаки/4task.py
ricardof@ox-i1 Числаки % /usr/local/bin/python3.12 /Users/ricardof/Desktop/Числаки/4task.py
Решение:
x = 0.057628481308925325
y = -0.4566222326825395
ricardof@ox-i1 Числаки %
```

Метод Ньютона:

$$x_{n+1} = x_n - J(x_n)^{-1} F(x_n)$$

$$\|x_{n+1} - x_n\| < \varepsilon$$

4btask.py > ...

```python
import numpy as np

def F(x):
    return np.array([
        2*x[0]-np.cos(x[1]+1),
        np.sin(x[0]) + 2*x[1] + 0.4
    ])

def J(x):
    return np.array([
        [2, np.sin(x[1]+1)],
        [np.cos(x[0]), 2]
    ])

def newton_system(F, J, x0):
    x = np.array(x0, dtype=float)
    while True:
        delta_x = np.linalg.solve(J(x), -F(x))
        x = x + delta_x
        if np.linalg.norm(delta_x) < e:
            break
    return x

x0 = [0.0, 0.0]
e = 0.001
solution = newton_system(F, J, x0)
print("Решение системы:", solution)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
/usr/local/bin/python3.12 /Users/ricardof/Desktop/Числаки/4btask.py
ricardof@ox-i1 Числаки % /usr/local/bin/python3.12 /Users/ricardof/Desktop/Числаки/4btask.py
Решение системы: [ 0.41283871 -0.40060558]
ricardof@ox-i1 Числаки %
```

5. Найти собственные значения матрицы: $A = \begin{pmatrix} 1 & 2 & \alpha \\ 2 & 3 & 4 \\ \alpha & 4 & 5 \end{pmatrix}$

8	$\alpha = -7;$

a. $x_{k+1} = A \cdot x_k$, где A - исходная матрица, а x_k - текущее приближение собственного вектора.

b. $\lambda_k = \frac{x_{k+1}^T \cdot x_k}{x_k^T \cdot x_k}$, где λ_k - текущее приближение собственного значения.

Нормализованный:

$$x_{k+1}: x_{k+1} = \frac{x_{k+1}}{\|x_{k+1}\|} \quad \mathbf{v}_{\text{нормализованный}} = \frac{\mathbf{v}}{\|\mathbf{v}\|}$$

5task.py > find_znach

```python
import numpy as np

def find_znach(A, num_iterations):
    n=3
    x = np.array([1,1,1])
    for _ in range(num_iterations):
        x = A.dot(x)
        x = x / (x[0]**2+x[1]**2+x[2]**2)
    sobstv_znach = np.dot(A.dot(x), x) / np.dot(x, x)
    return sobstv_znach

A = np.array([[1, 2, -7],
              [2, 3, 4],
              [-7, 4, 5]])

iterations = 1000
sobstv_znach = find_znach(A, iterations)
print("Собственное значение:", sobstv_znach)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
/usr/local/bin/python3.12 /Users/ricardof/Desktop/Числаки/5task.py
ricardof@ox-i1 Числаки % /usr/local/bin/python3.12 /Users/ricardof/Desktop/Числаки/5task.py
Собственное значение: 10.861711562867105
ricardof@ox-i1 Числаки %
```

6. По заданным значениям x и y найти прямую $y = a_0 + a_1 x$ и параболу $y = a_0 + a_1 x + a_2 x^2$ методом наименьших квадратов. Найти погрешность. Построить прямую и кривую в той же системе координат, где нанесены данные точки.

№ 8

N	1	2	3	4	5	6	7	8	9	10	11	12	13	14
X	0,40	0.,95	1,12	1,24	2,34	2,78	3,70	5,08	5,45	6,91	7,21	7,88	8,85	9,78
Y	0,96	2,23	2,38	2,98	4,77	6,07	7,77	10,32	11,68	14,17	14,34	16,05	17,72	20,10

Прямая:

Параметры прямой $y = mx + b$ - это наклон m и точка пересечения с осью y b.

Для прямой, метод наименьших квадратов сводится к вычислению следующих формул:

$$m = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2}$$

$$b = \bar{y} - m\bar{x}$$

Для аппроксимации данных параболой методом наименьших квадратов, мы можем воспользоваться нормальными уравнениями:

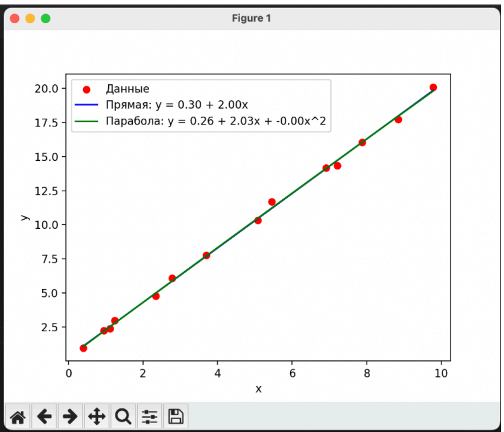
$$\begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \\ \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix}$$

Далее $A^T A\vec{a} = A^T\vec{y}$

Коэффициенты a_0, a_1 и a_2 находятся путем решения нормальных уравнений, что сводится к следующей матричной формуле:

$$\vec{a} = (A^T A)^{-1} A^T \vec{y}$$

```
6task.py ×
6task.py > ...
import numpy as np
import matplotlib.pyplot as plt
x = np.array([0.4, 0.95, 1.12, 1.24, 2.34, 2.78, 3.7, 5.08, 5.45, 6.91, 7.21, 7.88, 8.85, 9.78])
y = np.array([0.96, 2.23, 2.38, 2.98, 4.77, 6.07, 7.77, 10.32, 11.68, 14.17, 14.34, 16.05, 17.72, 20.1])
n = len(x)
sred_x = sum(x) / n
sred_y = sum(y) / n
chislitel = sum((x[i] - sred_x) * (y[i] - sred_y) for i in range(n))
znamenatel = sum((x[i] - sred_x) ** 2 for i in range(n))
a1_line = chislitel / znamenatel
a0_line = sred_y - a1_line * sred_x
A = np.vstack([np.ones(n), x, x**2]).T
C = np.linalg.lstsq(A, y, rcond=None)[0]
a0_parabola = C[0]
a1_parabola = C[1]
a2_parabola = C[2]
print("Параметры прямой:")
print(f"a0 = {a0_line}, a1 = {a1_line}")
print("Параметры параболы:")
print(f"a0 = {a0_parabola}, a1 = {a1_parabola}, a2 = {a2_parabola}")
plt.scatter(x, y, color='red', label='Данные')
y_line = a0_line + a1_line * x
plt.plot(x, y_line, color='blue', label='Прямая: y = {:.2f} + {:.2f}x'.format(a0_line, a1_line))
y_parabola = a0_parabola + a1_parabola * x + a2_parabola * x**2
plt.plot(x, y_parabola, color='green', label='Парабола: y = {:.2f} + {:.2f}x + {:.2f}x^2'.format(a0_parabola, a1_parabola, a2_parabola))
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```




```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   JUPYTER

/usr/local/bin/python3.12 /Users/ricardof/Desktop/Числаки/6task.py
ricardof@su-e5 Числаки % /usr/local/bin/python3.12 /Users/ricardof/Desktop/Числаки/6task.py
Параметры прямой:
a0 = 0.30142112259950515, a1 = 1.9990595742441033
Параметры параболы:
a0 = 0.2574037204192704, a1 = 2.0295296771395606, a2 = -0.0031379738107260025
```

7.

1) Заданы значения функции $f(x)$ в узлах x_i, получающиеся делением отрезка $[1,2]$ на 5 частей. Найти значения функции $f(x)$ при $x_1 = 1{,}1$ и $x_2 = 2{,}1$ с помощью интерполяционных формул Ньютона.

x_i	**8**
1,0	1,2
1,2	1,8
1,4	3,2
1,6	4,1
1,8	5,2
2,0	6,1

x	$f[x_0]$	$f[x_0,x_1]$	$f[x_0,x_1,x_2]$	$f[x_0,x_1,x_2,x_3]$	$f[x_0,x_1,x_2,x_3,x_4]$
1.0	1.2				
1.2	1.8	$\frac{1.8-1.2}{1.2-1.0}=3.0$			
1.4	3.2	$\frac{3.2-1.8}{1.4-1.2}=7.0$	$\frac{7.0-3.0}{1.4-1.0}=10.0$		
1.6	4.1	$\frac{4.1-3.2}{1.6-1.4}=4.5$	$\frac{4.5-7.0}{1.6-1.2}=-6.25$	$\frac{-6.25-10.0}{1.6-1.0}=-27.0833$	
1.8	5.2	$\frac{5.2-4.1}{1.8-1.6}=5.5$	$\frac{5.5-4.5}{1.8-1.4}=2.5$	$\frac{2.5-(-6.25)}{1.8-1.2}=14.5833$	$\frac{14.5833-(-27.0833)}{1.8-1.0}=53.1905$
2.0	6.1	$\frac{6.1-5.2}{2.0-1.8}=4.5$	$\frac{4.5-5.5}{2.0-1.6}=-2.5$	$\frac{-2.5-2.5}{2.0-1.4}=-8.3333$	$\frac{-8.3333-14.5833}{2.0-1.2}=-28.5926$

$$P_n(x) = f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + \cdots$$

7task.py > newton_interpolation

```python
def divided_diff(x, y):
    n = len(y)
    coef = [0] * n
    coef[0] = y[0]

    for j in range(1, n):
        for i in range(n - 1, j - 1, -1):
            y[i] = (y[i] - y[i - 1]) / (x[i] - x[i - j])
        coef[j] = y[j]
    return coef

def newton_interpolation(x, y, x_val):
    coef = divided_diff(x, y.copy())
    n = len(coef)
    result = coef[0]
    for i in range(1, n):
        term = coef[i]
        for j in range(i):
            term *= (x_val - x[j])
        result += term
    return result


x = [1.0, 1.2, 1.4, 1.6, 1.8, 2.0]
y = [1.2, 1.8, 3.2, 4.1, 5.2, 6.1]

x_vals = [1.1, 2.1]

results = {x_val: newton_interpolation(x, y, x_val) for x_val in x_vals}

for x_val, result in results.items():
    print(f"f({x_val}) = {result:.4f}")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
/usr/local/bin/python3.12 /Users/ricardof/Desktop/Числаки/7task.py
ricardof@ox-i1 Числаки % /usr/local/bin/python3.12 /Users/ricardof/Desktop/Числаки/7task.py
f(1.1) = 1.1559
f(2.1) = 5.2863
ricardof@ox-i1 Числаки %
```

2) Заданы значения y_i функции $f(x)$ в точках x_i. Найти значение функции $f(x)$ при $x = x^*$. Задачу решить с помощью интерполяционного многочлена Лагранжа.

8. Вычислить определённый интеграл с точностью $\varepsilon = 0{,}01$ методом Симпсона.

8	$\int_1^2 \frac{x+1}{2+\ln(1+x^2)}dx$	0,01

Интеграл от функции $f(x)$ на интервале от a до b с использованием метода Симпсона вычисляется по формуле:

$$\int_a^b f(x)\,dx \approx \frac{h}{3}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right]$$

где $h = \frac{b-a}{2}$.

8task.py > ...

```python
from math import *
def f(x):
    return (x+1)/(2+log(1+x**2))

def simpsons_rule(a, b):
    h = (b - a) / 2
    result = f(a) + f(b)

    for i in range(1, 2):
        if i % 2 == 0:
            result += 2 * f(a + i * h)
        else:
            result += 4 * f(a + i * h)

    result *= h / 3
    return result

def epsilon(a, b, e):
    n = 2
    integral_prev = 0
    integral_curr = simpsons_rule(a, b)

    while abs(integral_curr - integral_prev) > e:
        n *= 2
        integral_prev = integral_curr
        integral_curr = simpsons_rule(a, b)

    return integral_curr

a = 0
b = 1
e = 0.01

result = epsilon(a, b,e)
print("Результат вычисления определенного интеграла:", result)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

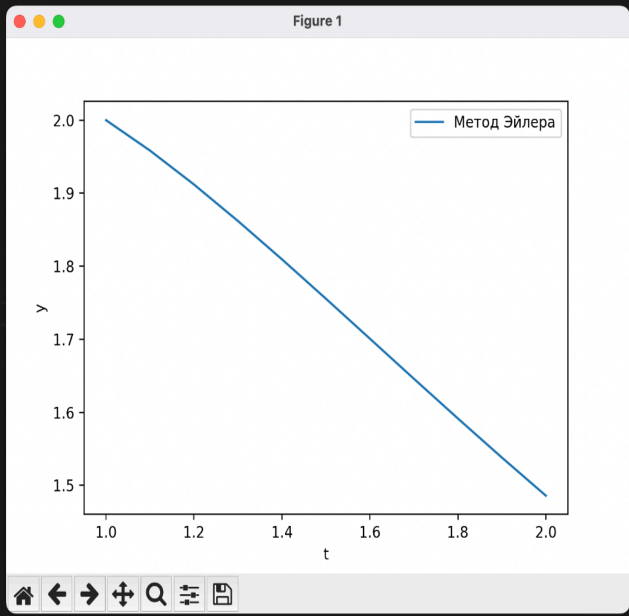
```
/usr/local/bin/python3.12 /Users/ricardof/Desktop/Числаки/8task.py
ricardof@su-e5 Числаки % /usr/local/bin/python3.12 /Users/ricardof/Desktop/Числаки/8task.py
Результат вычисления определенного интеграла: 0.65691777231657
ricardof@su-e5 Числаки %
```

9. Решить задачу Коши методом Эйлера и Рунге – Кутта.

8	$y'(t) = \cos\sqrt{ty^2}$	$y(0) = 2$	$[1,2]$	10

9task.py > ...

```python
import numpy as np
import matplotlib.pyplot as plt
from math import *

def eiler(f, t0, y0, t_end, h):
    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

def f(t, y):
    return cos((t*y**2)**(0.5))

t0 = 1
y0 = 2
T = 2
N = 0.1

t_values, y_values = eiler(f, t0, y0, T, N)

plt.plot(t_values, y_values, label='Метод Эйлера')
plt.xlabel('t')
plt.ylabel('y')
plt.legend()
plt.show()
```



PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   JUPYTER

```
/usr/local/bin/python3.12 /Users/ricardof/Desktop/Числаки/8task.py
ricardof@su-e5 Числаки % /usr/local/bin/python3.12 /Users/ricardof/Desktop/Числаки/8task.py
Результат вычисления определенного интеграла: 0.65691777231657
ricardof@su-e5 Числаки % /usr/local/bin/python3.12 /Users/ricardof/Desktop/Числаки/9task.py
ricardof@su-e5 Числаки % /usr/local/bin/python3.12 /Users/ricardof/Desktop/Числаки/9task.py
ricardof@su-e5 Числаки % /usr/local/bin/python3.12 /Users/ricardof/Desktop/Числаки/9task.py
/usr/local/bin/python3.12 /Users/ricardof/Desktop/Числаки/9task.py
/usr/local/bin/python3.12 /Users/ricardof/Desktop/Числаки/9task.py
ricardof@su-e5 Числаки % /usr/local/bin/python3.12 /Users/ricardof/Desktop/Числаки/9task.py
ricardof@su-e5 Числаки % /usr/local/bin/python3.12 /Users/ricardof/Desktop/Числаки/9task.py
ricardof@su-e5 Числаки % /usr/local/bin/python3.12 /Users/ricardof/Desktop/Числаки/9task.py
```