

# РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра Институт Математики

## ОТЧЕТ ПО ЗАДАНИЮ № 2

Дисциплина: Дополнительные главы вычислительных методов

Студент: Доре Стевенсон Эдгар

Группа: НПМ-01-23

МОСКВА

2022 г.

## Содержание

Теоретическая справка.....	
Программный код.....	
Численные расчеты.....	
Заключение .....	

### Краткая теоретическая справка

**Цель:** Реализовать в программе метод решения системы линейных алгебраических уравнений (СЛАУ) на основе метода Якоби .

**Краткое описание:**

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n &= f_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n &= f_2 \\ &\dots \\ a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n &= f_n \end{aligned} \tag{1}$$

или в сокращённой записи:

$$\sum_{j=1}^n a_{ij}x_j = f_i, i = 1, 2, \dots, n.$$

Коэффициенты  $a_{ij}$  при неизвестных  $x_j$  образуют матрицу системы:

$$A = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nn} \end{bmatrix} \tag{2}$$

Всюду на протяжении этой лабораторной работы будем считать определитель матрицы отличным от нуля:

$$\Delta = \det A \neq 0. \tag{3}$$

В этом случае система называется невырожденной.

Для решения задач стоят два метода : **Прямой и Интеграции**

Прямой метод предполагает решение одного уравнения за один раз путем исключения одной переменной из каждого уравнения до тех пор, пока все переменные не будут решены. Этот метод хорошо работает, если уравнений столько же, сколько неизвестных (переменных), но он усложняется, если уравнений меньше, чем неизвестных. В этом случае для создания новых уравнений могут быть использованы дополнительные методы, такие как

гауссова элиминация.

С другой стороны, в итерационном методе для поиска решений используются методы численной аппроксимации. Один из распространенных подходов называется итерацией Гаусса-Зейделя, которая решает задачу для одной переменной, сохраняя остальные фиксированными, а затем обновляет их значения, используя новое найденное решение.

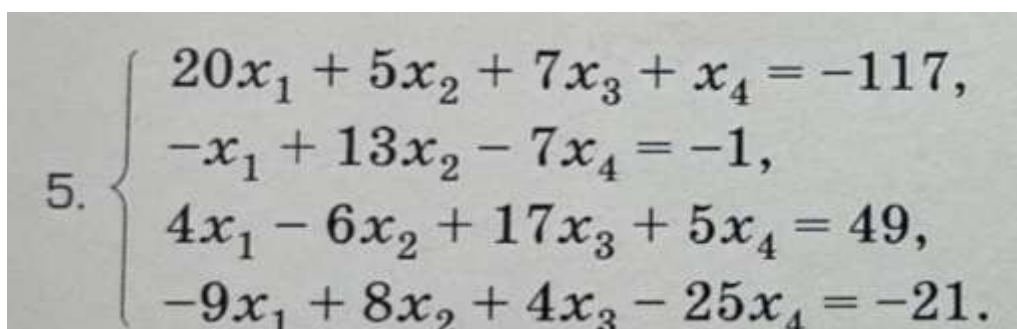
Другой подход - итерация Якоби, при которой все переменные обновляются одновременно на основе их текущих оценок.

### Итерационный метод Якоби

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left( b_i - \sum_{j \neq i} a_{ij} x_j^{(k)} \right), \quad i = 1, 2, \dots, n.$$

### Ход работы

#### Вариант 5


$$5. \begin{cases} 20x_1 + 5x_2 + 7x_3 + x_4 = -117, \\ -x_1 + 13x_2 - 7x_4 = -1, \\ 4x_1 - 6x_2 + 17x_3 + 5x_4 = 49, \\ -9x_1 + 8x_2 + 4x_3 - 25x_4 = -21. \end{cases}$$

Программный код реализован на языке Python

Программный код с подробными комментариями

```

In [2]: # Define the Jacobi method function
def jacobi(A, b, a=0.0001, max_iter=100):
    # Get the number of rows in the coefficient matrix A
    n = len(A)
    # Initialize the solution vector x to all zeros
    x = np.zeros(n)
    # Initialize the previous solution vector x_prev to all zeros
    x_prev = np.zeros(n)
    # Iterate until the maximum number of iterations is reached
    for i in range(max_iter):
        # Iterate over each row of the coefficient matrix A
        for j in range(n):
            # Compute the updated value of the jth component of the solution vector x
            x[j] = (b[j] - np.dot(A[j,:], x_prev) + A[j,j]*x_prev[j]) / A[j,j]
        # Compute the error estimate between the current and previous solution vectors
        e = np.max(np.abs(x - x_prev))
        # Check if the error estimate is less than the specified tolerance
        if e < a:
            # If the error estimate is less than the specified tolerance, return the solution vector x
            return x
        # Update the previous solution vector x_prev to the current solution vector x
        x_prev = np.copy(x)
    # If the maximum number of iterations is reached, return the solution vector x
    return x

In [4]: # Define the coefficient matrix A and the right-hand side vector b
A = np.array([[20, 5, 7, 1], [-1, 13, -7, -7], [4, 6, 17, 5], [9, 8, 14, -25]])
b = np.array([-117, -1, 49, -21])
# Solve the linear system of equations using the Jacobi method with a tolerance of 0.0001 and a maximum of 100 iterations
x = jacobi(A, b, a=0.0001, max_iter=100)
# Print the solution vector x
print(x)

```

## Численные расчеты

---

`[-7.67864496 1.81403044 3.81562557 0.79297299]`

Мы определяем функцию Якоби, которая принимает на вход матрицу коэффициентов  $A$ , вектор правой части  $b$ , допуск на относительную ошибку  $a$  и максимальное количество итераций  $\text{max\_iter}$ . Функция инициализирует вектор решения  $x$  всеми нулями, а затем итеративно обновляет  $x$  по методу Якоби до тех пор, пока оценка ошибки не станет меньше заданного допуска или не будет достигнуто максимальное количество итераций. Функция возвращает окончательный вектор решения  $x$ .

Далее определяем матрицу коэффициентов  $A$  размером  $4 \times 4$  и 4-элементный вектор правой части  $b$ , вызываем функцию Якоби с этими входными данными, задаем допуск на относительную ошибку  $0,0001$  и максимальное количество итераций  $100$ .

Функция возвращает вектор решений  $x$ , который выводится на консоль.

### Задание 1 :

Цель : Построить 3 D график функции и значения функции на краях

Краткое описания :

Handwritten mathematical notes on a piece of paper:

- $x^2 - y^2 = f(x, y)$
- $[0, 1] \times [0, 1]$
- $N = 50$
- $h = \frac{b-a}{N}$
- $x_i = x_0 + i \cdot h$
- $y_i = y_0 + j \cdot h$
- A bracketed list of values:
  - $f(0, y)$
  - $f(x, 0)$
  - $f(1, y)$
  - $f(x, 1)$
- At the bottom right, a small table structure is visible:

$x_i$	$y_i$	$f(x_i, y_i)$
$\vdots$	$\vdots$	$\vdots$

Код программирования на Python

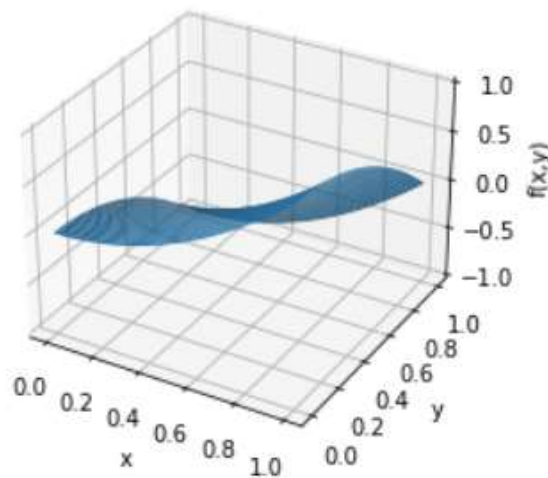
```
In [2]: import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

```
In [3]: # Define the function  $f(x,y) = x^2 - y^2$ 
def f(x, y):
    return x**2 - y**2
```

```
In [4]: # Create a 3D grid of points in the domain  $[0,1]*[0,1]$ 
x = np.linspace(0, 1, 50)
y = np.linspace(0, 1, 50)
X, Y = np.meshgrid(x, y)
```

```
In [5]: # Evaluate the function  $f(x,y)$  at each point in the grid
Z = f(X, Y)
```

```
In [9]: # Create a 3D plot of the function using Matplotlib's mplot3d toolkit
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, Z)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('f(x,y)')
plt.show()
```



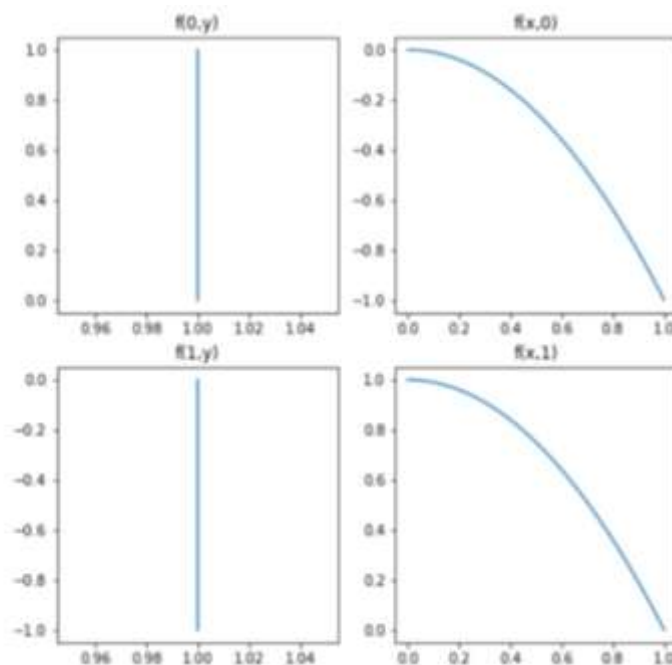
```
In [10]: # Evaluate the function f(x,y) at the edges of the domain [0,1]*[0,1]
x = np.linspace(0, 1, 50)
y = np.zeros(50)
f1 = f(x, y)
```

```
In [11]: x = np.zeros(50)
y = np.linspace(0, 1, 50)
f2 = f(x, y)
```

```
In [12]: x = np.linspace(0, 1, 50)
y = np.ones(50)
f3 = f(x, y)
```

```
In [13]: x = np.ones(50)
y = np.linspace(0, 1, 50)
f4 = f(x, y)
```

```
In [14]: # Create 2D plots of the function f(x,y) at the edges of the domain
fig, axs = plt.subplots(2, 2, figsize=(8, 8))
axs[0, 0].plot(x, f1)
axs[0, 0].set_title('f(0,y)')
axs[0, 1].plot(y, f2)
axs[0, 1].set_title('f(x,0)')
axs[1, 0].plot(x, f3)
axs[1, 0].set_title('f(1,y)')
axs[1, 1].plot(y, f4)
axs[1, 1].set_title('f(x,1)')
plt.show()
```





Заключение :

1. Мы Реализовали в программе метод решения системы линейных алгебраических уравнений (СЛАУ) на основе метода Якоби .
2. Мы построили 3 D график функции и ее значения на краях