

## Task1:

The screenshot shows the PyCharm IDE with a project named 'pythonProject1'. The file explorer on the left shows a directory structure with 'tasks.py' and several other task files. The main editor displays the code for 'tasks.py', which implements the Jacobi method. The code defines a function 'jacobi\_method(A, b, x0, tol=1e-5, max\_iterations=100)' that iteratively solves the system  $AX = b$  using the Jacobi iteration formula. It also includes a function 'is\_diagonally\_dominant(A)' to check if the matrix is diagonally dominant. The console output shows the results of running the code: 'Diagonal dominance: True', 'Solution: [ 0.77443213 -0.293237 1.51879306]', 'Iterations: 9', and 'Convergence Explanation: The system converges because the spectral radius is less than 1.' The status bar at the bottom indicates the file encoding is UTF-8, the line length is 131, and the Python version is 3.12.

```
def jacobi_method(A, b, x0, tol=1e-5, max_iterations=100):
    n = len(b)
    x = np.array(x0, dtype=float)
    x_new = np.zeros_like(x)
    iterations = 0

    for _ in range(max_iterations):
        for i in range(n):
            s = sum(A[i][j] * x[j] for j in range(n) if j != i)
            x_new[i] = (b[i] - s) / A[i][i]

        # Check for convergence
        if np.linalg.norm(x_new - x, ord=np.inf) < tol:
            break

        x[:] = x_new
        iterations += 1

    return x_new, iterations

def is_diagonally_dominant(A):
    for i in range(len(A)):
        row_sum = sum(abs(A[i][j]) for j in range(len(A)) if j != i)
        if row_sum >= abs(A[i][i]):
            return False
    return True

# Example usage
A = np.array([[4, 1, 1], [1, 4, 1], [1, 1, 4]], dtype=float)
b = np.array([14, 14, 14], dtype=float)
x0 = np.zeros(3)
x, iterations = jacobi_method(A, b, x0)
print("Diagonal dominance: True")
print("Solution: ", x)
print("Iterations: ", iterations)
print("Convergence Explanation: The system converges because the spectral radius is less than 1.")
```

## Task2:

The screenshot shows the PyCharm IDE with a project named 'pythonProject1'. The file explorer on the left shows a directory structure with 'tasks.py' and several other task files. The main editor displays the code for 'tasks.py', which implements Gaussian elimination with pivoting. The code defines a function 'gaussian\_elimination\_with\_pivoting(A, b)' that performs row operations to transform the matrix A into an upper triangular form. It then uses back-substitution to find the solution x. The console output shows the results of running the code: 'Diagonal dominance: True', 'Solution: [ 0.77443213 -0.293237 1.51879306]', 'Iterations: 9', and 'Convergence Explanation: The system converges because the spectral radius is less than 1.' The status bar at the bottom indicates the file encoding is UTF-8, the line length is 84, and the Python version is 3.12.

```
def gaussian_elimination_with_pivoting(A, b):
    n = len(A)
    # Прямой ход с выбором ведущего элемента
    for k in range(n):
        # Находим строку с максимальным ведущим элементом
        max_row = max(range(k, n), key=lambda i: abs(A[i][k]))

        # Меняем строки местами
        if max_row != k:
            A[[k, max_row]] = A[[max_row, k]]
            b[[k, max_row]] = b[[max_row, k]]

        # Приводим матрицу к верхнетреугольному виду
        for i in range(k + 1, n):
            factor = A[i][k] / A[k][k]
            A[i, k:] -= factor * A[k, k:]
            b[i] -= factor * b[k]

    # Обратный ход для нахождения решения
    x = np.zeros(n)
    for i in range(n - 1, -1, -1):
        x[i] = (b[i] - np.dot(A[i, i + 1:], x[i + 1:])) / A[i, i]

    return A, x

# Заданная система уравнений
A = np.array([[2, 3, 1], [4, 11, -1], [-2, 1, 7]], dtype=float)
b = np.array([10, 33, 15], dtype=float)
```

```

C:\Users\bookn\PycharmProjects\pythonPr
Верхнетреугольная матрица:
[[ 4.  11. -1. ]
 [ 0.   6.5 6.5]
 [ 0.   0.  4. ]]

Решение системы:
[-0.86538462  3.44230769  1.40384615]
Верхнетреугольная матрица:
[[ 4.  11. -1. ]
 [ 0.   6.5 6.5]
 [ 0.   0.  4. ]]

Решение системы:
[-0.86538462  3.44230769  1.40384615]

```

### Task3:

```

3 def gauss_jordan(a):
4     n = len(a)
5     for i in range(n):
6         # Поиск максимального элемента в столбце для выбора ведущего элемента
7         max_row = i + np.argmax(np.abs(a[i:, i]))
8         a[[i, max_row]] = a[[max_row, i]] # Меняем строки местами
9
10        # Нормализация текущей строки
11        a[i] = a[i] / a[i, i]
12
13        # Обнуление элементов столбца, кроме ведущего
14        for j in range(n):
15            if i != j:
16                a[j] = a[j] - a[j, i] * a[i]
17
18    return a
19
20 # Коэффициенты системы
21 A = np.array([
22     [1, 1, 1, 9],
23     [2, -3, 4, 13],
24     [3, 4, 5, 40]
25 ], dtype=float)
26
27 # Решение методом Гаусса-Жордана
28 result = gauss_jordan(A)
29
30 # Вывод диагональной матрицы и решений
31 print("Диагональная матрица:")
32 print(result)
33
34 solutions = result[:, -1]
35 print("Решения переменных:")

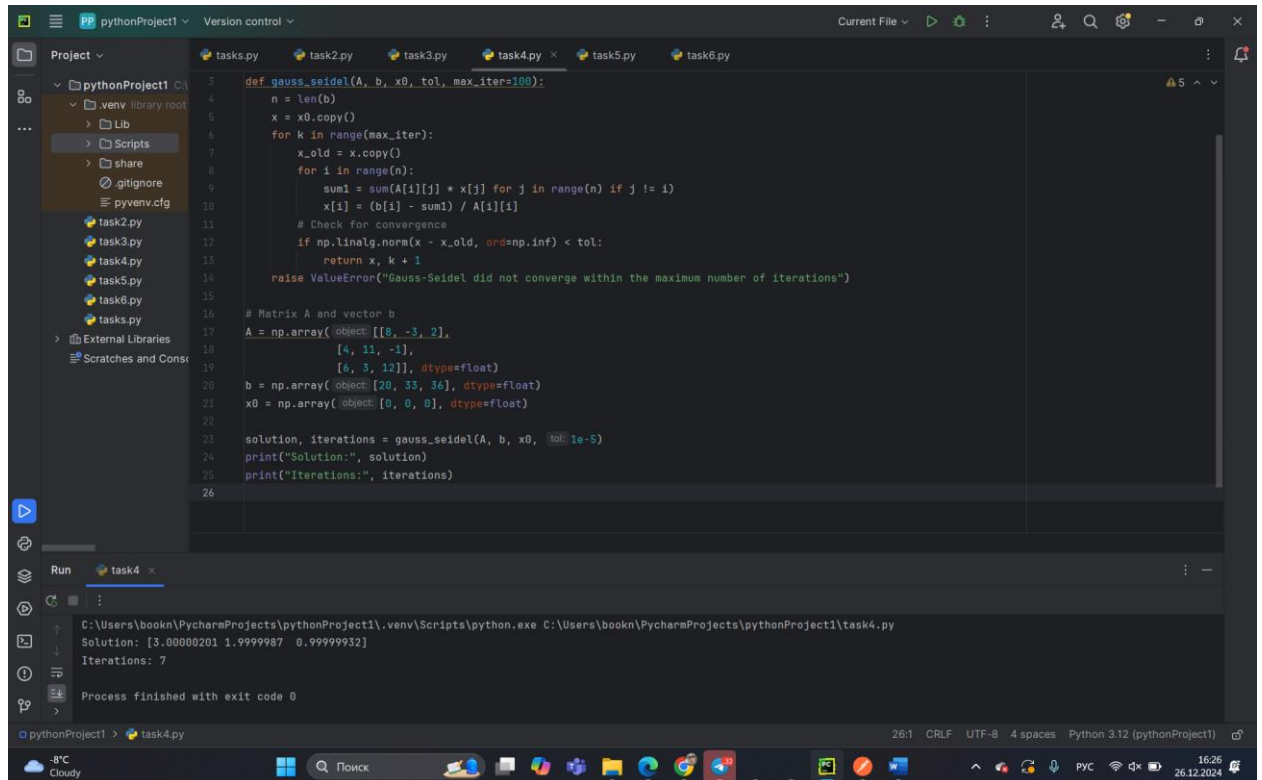
```

```

C:\Users\bookn\PycharmProjects\pythonProject1\.venv\Scripts\python.exe
Диагональная матрица:
[[ 1.  0.  0.  1.]
 [-0.  1.  0.  3.]
 [-0. -0.  1.  5.]]
Решения переменных:
x = 0.9999999999999982, y = 3.0000000000000004, z = 5.000000000000001
Process finished with exit code 0

```

## Task4:



```
def gauss_seidel(A, b, x0, tol, max_iter=100):
    n = len(b)
    x = x0.copy()
    for k in range(max_iter):
        x_old = x.copy()
        for i in range(n):
            sum1 = sum(A[i][j] * x[j] for j in range(n) if j != i)
            x[i] = (b[i] - sum1) / A[i][i]
        # Check for convergence
        if np.linalg.norm(x - x_old, ord=np.inf) < tol:
            return x, k + 1
    raise ValueError("Gauss-Seidel did not converge within the maximum number of iterations")

# Matrix A and vector b
A = np.array([
    [8, -3, 2],
    [4, 11, -1],
    [6, 3, 12]
], dtype=float)
b = np.array([20, 33, 36], dtype=float)
x0 = np.array([0, 0, 0], dtype=float)

solution, iterations = gauss_seidel(A, b, x0, tol=1e-5)
print("Solution:", solution)
print("Iterations:", iterations)
```

Run task4

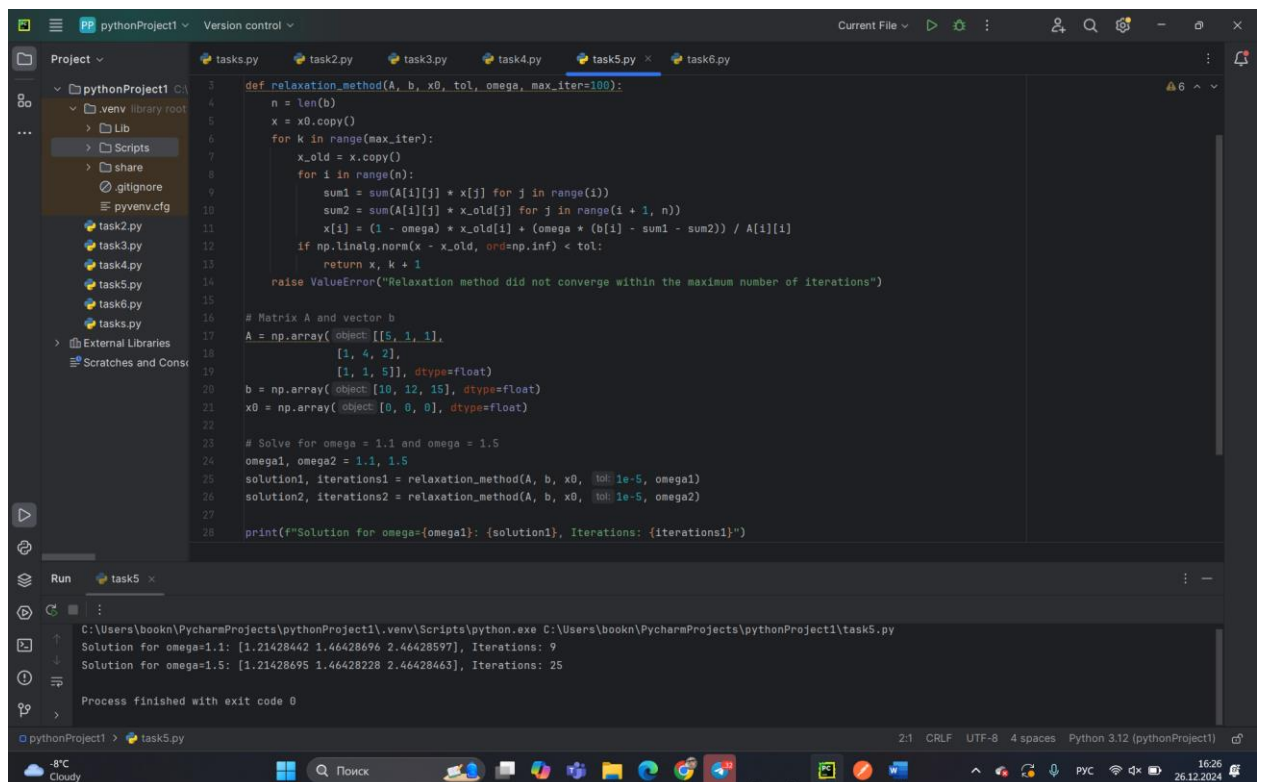
C:\Users\bookn\PycharmProjects\pythonProject1\.venv\Scripts\python.exe C:\Users\bookn\PycharmProjects\pythonProject1\task4.py

Solution: [3.00000201 1.99999987 0.99999932]

Iterations: 7

Process finished with exit code 0

## Task5:



```
def relaxation_method(A, b, x0, tol, omega, max_iter=100):
    n = len(b)
    x = x0.copy()
    for k in range(max_iter):
        x_old = x.copy()
        for i in range(n):
            sum1 = sum(A[i][j] * x[j] for j in range(i))
            sum2 = sum(A[i][j] * x_old[j] for j in range(i + 1, n))
            x[i] = (1 - omega) * x_old[i] + (omega * (b[i] - sum1 - sum2)) / A[i][i]
        if np.linalg.norm(x - x_old, ord=np.inf) < tol:
            return x, k + 1
    raise ValueError("Relaxation method did not converge within the maximum number of iterations")

# Matrix A and vector b
A = np.array([
    [5, 1, 1],
    [1, 4, 2],
    [1, 1, 5]
], dtype=float)
b = np.array([10, 12, 15], dtype=float)
x0 = np.array([0, 0, 0], dtype=float)

# Solve for omega = 1.1 and omega = 1.5
omega1, omega2 = 1.1, 1.5
solution1, iterations1 = relaxation_method(A, b, x0, tol=1e-5, omega=omega1)
solution2, iterations2 = relaxation_method(A, b, x0, tol=1e-5, omega=omega2)

print(f"Solution for omega={omega1}: {solution1}, Iterations: {iterations1}")
```

Run task5

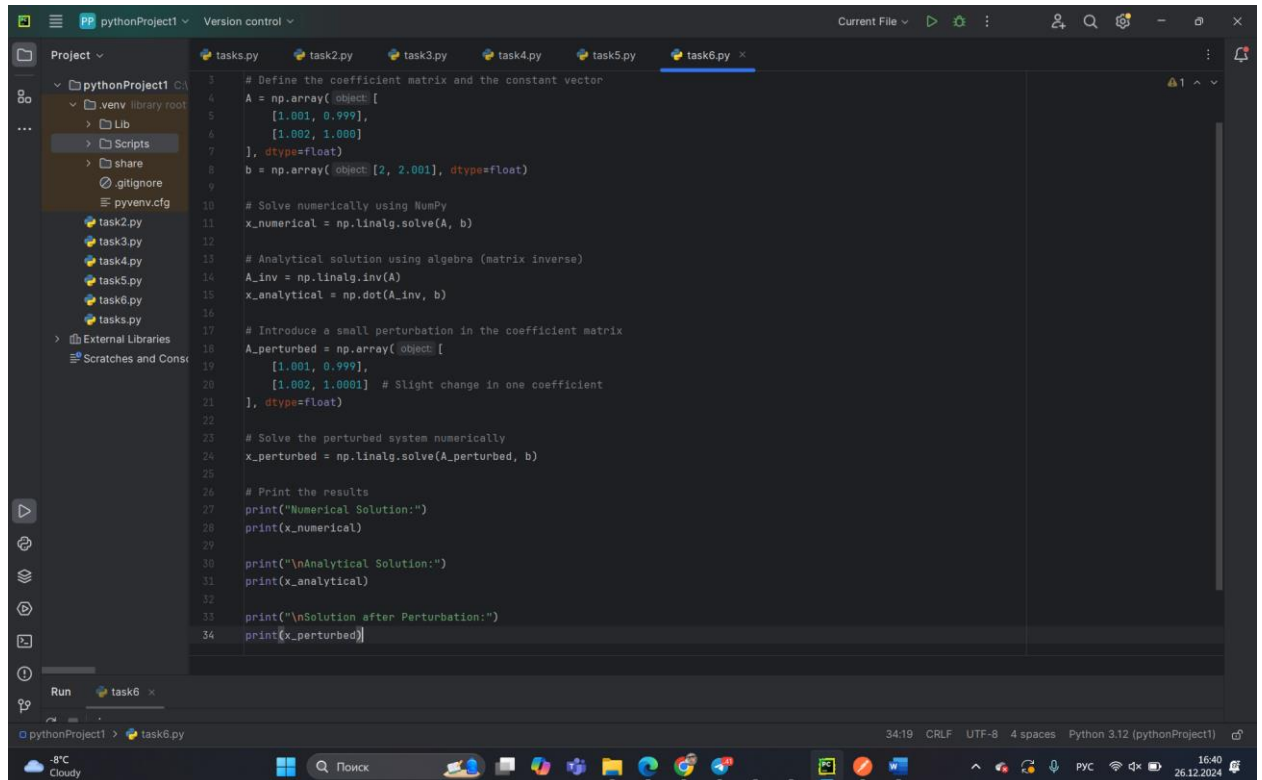
C:\Users\bookn\PycharmProjects\pythonProject1\.venv\Scripts\python.exe C:\Users\bookn\PycharmProjects\pythonProject1\task5.py

Solution for omega=1.1: [1.21428442 1.46428696 2.46428597], Iterations: 9

Solution for omega=1.5: [1.21428695 1.46428228 2.46428463], Iterations: 25

Process finished with exit code 0

## Task6:



The screenshot shows the PyCharm IDE with a project named 'pythonProject1'. The file explorer on the left shows a directory structure with files like 'tasks.py', 'task2.py', 'task3.py', 'task4.py', 'task5.py', and 'task6.py'. The main editor window displays the code for 'task6.py'. The code defines a coefficient matrix A and a constant vector b, solves the system numerically using NumPy's linalg.solve, and also calculates an analytical solution using matrix inversion. It then introduces a small perturbation to the coefficient matrix A, solves the perturbed system numerically, and prints the results for both the original and perturbed systems.

```
1 # Define the coefficient matrix and the constant vector
2
3 A = np.array([
4     [1.001, 0.999],
5     [1.002, 1.000]
6 ], dtype=float)
7
8 b = np.array([2, 2.001], dtype=float)
9
10 # Solve numerically using NumPy
11 x_numerical = np.linalg.solve(A, b)
12
13 # Analytical solution using algebra (matrix inverse)
14 A_inv = np.linalg.inv(A)
15 x_analytical = np.dot(A_inv, b)
16
17 # Introduce a small perturbation in the coefficient matrix
18 A_perturbed = np.array([
19     [1.001, 0.999],
20     [1.002, 1.0001] # Slight change in one coefficient
21 ], dtype=float)
22
23 # Solve the perturbed system numerically
24 x_perturbed = np.linalg.solve(A_perturbed, b)
25
26 # Print the results
27 print("Numerical Solution:")
28 print(x_numerical)
29
30 print("\nAnalytical Solution:")
31 print(x_analytical)
32
33 print("\nSolution after Perturbation:")
34 print(x_perturbed)
```

```
C:\Users\bookn\PycharmProjects\pythonProj
Numerical Solution:
[ 500.50000003 -499.50000003]

Analytical Solution:
[ 500.50000003 -499.50000003]

Solution after Perturbation:
[11.76297747 -9.78452498]

Process finished with exit code 0
```