Name: Phan Tran Thanh Huy

ID: ITCSIU22056

Lab 3

Activity 1: Gaussian Elimination

Code:

```python
import numpy as np

def f(A, b, x):
    return np.dot(A, x) - b

def forward_elimination(A, b):
    n = len(A)
    for i in range(n):
        max_row = max(range(i, n), key=lambda r: abs(A[r][i]))
        if i != max_row:
            A[[i, max_row]] = A[[max_row, i]]
            b[[i, max_row]] = b[[max_row, i]]

        for j in range(i+1, n):
            factor = A[j][i] / A[i][i]
            A[j] -= factor * A[i]
            b[j] -= factor * b[i]
    return A, b

def backward_substitution(A, b):
    n = len(A)
    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 x

def gaussian_elimination(A, b):
    A, b = forward_elimination(A, b)
    return backward_substitution(A, b)

def solve_linear_system():
    A = np.array([[3, 1, -2], [2, -2, 4], [-1, 12, -1]], dtype=float)
    b = np.array([1, -2, 0], dtype=float)
    solution = gaussian_elimination(A, b)
    print(f"\nApproximate solution:\n x = {solution[0]}\n y = {solution[1]}\n z = {solution[2]}")

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

Result:

```
PS E:\Homework\TMC\Lab3> & C:/Python312/python.exe e:/Homework/TMC/Lab3/p1.py

Approximate solution:
 x = 0.0
 y = -0.043478260869565
 z = -0.5217391304347826
```

Part 2: Iterative Methods (Jacobi and Gauss-Seidel)

Code:

```python
import numpy as np

def jacobi_method(A, b, tol=1e-6, max_iter=100):
    n = len(A)
    x = np.zeros(n)
    x_new = np.zeros(n)
    iter_count = 0

    print(f"{'Iter':<5}{'x':<15}{'y':<15}{'z':<15}")
    print("-" * 60)

    for _ in range(max_iter):
        for i in range(n):
            x_new[i] = (b[i] - np.sum(A[i, :i] * x[:i]) - np.sum(A[i, i+1:] * x[i+1:])) / A[i, i]

        error = np.linalg.norm(x_new - x, ord=np.inf)
        if iter_count < 3:
            print(f"{iter_count:<5}{x_new[0]:<15.6f}{x_new[1]:<15.6f}{x_new[2]:<15.6f}")

        if error < tol:
            return x_new
        x = x_new.copy()
        iter_count += 1

    return x_new

def gauss_seidel_method(A, b, tol=1e-6, max_iter=100):
    n = len(A)
    x = np.zeros(n)
    iter_count = 0

    print(f"{'Iter':<5}{'x':<15}{'y':<15}{'z':<15}")
    print("-" * 60)

    for _ in range(max_iter):
        x_old = x.copy()
        for i in range(n):
            x[i] = (b[i] - np.sum(A[i, :i] * x[:i]) - np.sum(A[i, i+1:] * x[i+1:])) / A[i, i]

        error = np.linalg.norm(x - x_old, ord=np.inf)
        if iter_count < 3:
            print(f"{iter_count:<5}{x[0]:<15.6f}{x[1]:<15.6f}{x[2]:<15.6f}")

        if error < tol:
            return x
        iter_count += 1

    return x

def solve_iterative_methods():
    A = np.array([[5, -2, 3], [2, 5, -1], [1, 3, 5]], dtype=float)
    b = np.array([10, 4, 8], dtype=float)

    print("\nSolving using Jacobi Method:")
    jacobi_solution = jacobi_method(A, b)
    print(f"\nJacobi Solution in 100th approximation:\n x = {jacobi_solution[0]}\n y = {jacobi_solution[1]}\n z = {jacobi_solution[2]}")

    print("\nSolving using Gauss-Seidel Method:")
    gauss_seidel_solution = gauss_seidel_method(A, b)
    print(f"\nGauss-Seidel Solution in 100th approximation:\n x = {gauss_seidel_solution[0]}\n y = {gauss_seidel_solution[1]}\n z = {gauss_seidel_solution[2]}")

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

Result:

```
Solving using Jacobi Method:
Iter x              y              z
------------------------------------------------------------
0    2.000000       0.800000       1.600000
1    1.360000       0.320000       0.720000
2    1.696000       0.400000       1.136000

Jacobi Solution in 100th approximation:
 x = 1.5272722970802626
 y = 0.40000000337761177
 z = 1.0545458454082763
```

```
Solving using Gauss-Seidel Method:
Iter x              y              z
------------------------------------------------------------
0    2.000000       0.000000       1.200000
1    1.280000       0.528000       1.027200
2    1.594880       0.367488       1.060531

Gauss-Seidel Solution:
 x = 1.5272725635955404
 y = 0.4000000767312931
 z = 1.0545454412421162
```

Part 3: Comparative Analysis

Code:

```python
import numpy as np
import time

def gaussian_elimination(A, b):
    start_time = time.time()
    A = A.astype(float)
    b = b.astype(float)
    n = len(A)

    for i in range(n):
        max_row = max(range(i, n), key=lambda r: abs(A[r][i]))
        if i != max_row:
            A[[i, max_row]] = A[[max_row, i]]
            b[[i, max_row]] = b[[max_row, i]]

        for j in range(i+1, n):
            factor = A[j][i] / A[i][i]
            A[j] -= factor * A[i]
            b[j] -= factor * b[i]

    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]

    end_time = time.time()
    return x, end_time - start_time, 1

def jacobi_method(A, b, tol=1e-6, max_iter=100):
    start_time = time.time()
    n = len(A)
    x = np.zeros(n)
    x_new = np.zeros(n)
    iter_count = 0

    for _ in range(max_iter):
        for i in range(n):
            x_new[i] = (b[i] - np.sum(A[i, :i] * x[:i]) - np.sum(A[i, i+1:] * x[i+1:])) / A[i, i]

        error = np.linalg.norm(x_new - x, ord=np.inf)
        if error < tol:
            end_time = time.time()
            return x_new, end_time - start_time, iter_count
        x = x_new.copy()
        iter_count += 1

    end_time = time.time()
    return x_new, end_time - start_time, iter_count

def gauss_seidel_method(A, b, tol=1e-6, max_iter=100):
    start_time = time.time()
    n = len(A)
    x = np.zeros(n)
    iter_count = 0

    for _ in range(max_iter):
        x_old = x.copy()
        for i in range(n):
            x[i] = (b[i] - np.sum(A[i, :i] * x[:i]) - np.sum(A[i, i+1:] * x[i+1:])) / A[i, i]

        error = np.linalg.norm(x - x_old, ord=np.inf)
        if error < tol:
            end_time = time.time()
            return x, end_time - start_time, iter_count
        iter_count += 1

    end_time = time.time()
    return x, end_time - start_time, iter_count

def solve_and_compare_methods():
    A = np.array([[3, 1, -2], [2, -2, 4], [-1, 12, -1]], dtype=float)
    b = np.array([1, -2, 0], dtype=float)

    ge_solution, ge_time, ge_iters = gaussian_elimination(A.copy(), b.copy())
    jacobi_solution, jacobi_time, jacobi_iters = jacobi_method(A.copy(), b.copy())
    gs_solution, gs_time, gs_iters = gauss_seidel_method(A.copy(), b.copy())

    print("\nComparison of Methods:\n")
    print(f"{'Method':<15}{'x':<15}{'y':<15}{'z':<15}{'Time (s)':<15}{'Iterations':<10}")
    print("-" * 80)
    print(f"{'Gaussian':<15}{ge_solution[0]:<15.6g}{ge_solution[1]:<15.6g}{ge_solution[2]:<15.6g}{ge_time:<15.6f}{'-':<10}")
    print(f"{'Jacobi':<15}{jacobi_solution[0]:<15.6g}{jacobi_solution[1]:<15.6g}{jacobi_solution[2]:<15.6g}{jacobi_time:<15.6f}{jacobi_iters:<10}")
    print(f"{'Gauss-Seidel':<15}{gs_solution[0]:<15.6g}{gs_solution[1]:<15.6g}{gs_solution[2]:<15.6g}{gs_time:<15.6f}{gs_iters:<10}")

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

Result:

```
Comparison of Methods:

Method        x             y             z             Time (s)      Iterations
---------------------------------------------------------------------------
Gaussian      0             -0.0434783    -0.521739     0.000000      -
Jacobi        1.52038e+68   6.09249e+68   1.43994e+69   0.002536      100
Gauss-Seidel  1.57464e+147  6.50836e+147  7.65257e+148  0.002999      100
```

Discuss:

Method	**Advantages**	**Disadvantages**
Gaussian Elimination	Direct, fast for small systems, works for any matrix	Computationally expensive for large matrices ($O(n^3)$), prone to round-off errors.
Jacobi Method	Simple, easy to implement, parallelizable	Slow convergence, needs diagonal dominance.
Gauss-Seidel Method	Faster than Jacobi, fewer iterations	Still slower than direct methods, needs diagonal dominance.

Part 4: Exercise

Code:

```python
import numpy as np

def jacobi_method(A, b, tol=1e-6, max_iter=100):
    n = len(A)
    x = np.zeros(n)
    x_new = np.zeros(n)
    iter_count = 0

    print(f"\n{'Jacobi Method':^50}")
    print(f"{'Iter':<5}{'x':<15}{'y':<15}{'z':<15}")
    print("-" * 50)

    for _ in range(max_iter):
        for i in range(n):
            x_new[i] = (b[i] - np.sum(A[i, :i] * x[:i]) - np.sum(A[i, i+1:] * x[i+1:])) / A[i, i]

        error = np.linalg.norm(x_new - x, ord=np.inf)
        print(f"{iter_count:<5}{x_new[0]:<15.6f}{x_new[1]:<15.6f}{x_new[2]:<15.6f}")

        if error < tol:
            return x_new, iter_count
        x = x_new.copy()
        iter_count += 1

    return x_new, iter_count

def gauss_seidel_method(A, b, tol=1e-6, max_iter=100):
    n = len(A)
    x = np.zeros(n)
    iter_count = 0

    print(f"\n{'Gauss-Seidel Method':^50}")
    print(f"{'Iter':<5}{'x':<15}{'y':<15}{'z':<15}")
    print("-" * 50)

    for _ in range(max_iter):
        x_old = x.copy()
        for i in range(n):
            x[i] = (b[i] - np.sum(A[i, :i] * x[:i]) - np.sum(A[i, i+1:] * x[i+1:])) / A[i, i]

        error = np.linalg.norm(x - x_old, ord=np.inf)
        print(f"{iter_count:<5}{x[0]:<15.6f}{x[1]:<15.6f}{x[2]:<15.6f}")

        if error < tol:
            return x, iter_count
        iter_count += 1

    return x, iter_count

def solve_iterative_methods():
    A = np.array([[5, -2, 3], [2, 5, -1], [1, 3, 5]], dtype=float)
    b = np.array([10, 4, 8], dtype=float)

    jacobi_solution, jacobi_iters = jacobi_method(A, b)
    gauss_seidel_solution, gauss_seidel_iters = gauss_seidel_method(A, b)

    print("\nFinal Comparison:")
    print(f"{'Method':<15}{'Iterations':<10}")
    print("-" * 30)
    print(f"{'Jacobi':<15}{jacobi_iters:<10}")
    print(f"{'Gauss-Seidel':<15}{gauss_seidel_iters:<10}")

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

Result:

```
PS E:\Homework\TMC\Lab3> & C:/Python312/python.exe e:/Homework/TMC/Lab3/p4.py
Jacobi Method achieved in 25 iterations.
Solution: x = 1.527272, y = 0.400000, z = 1.054546

Gauss-Seidel Method achieved in 11 iterations.
Solution: x = 1.527273, y = 0.400000, z = 1.054545
```