

Name: Phan Tran Thanh Huy

ID: ITCSIU22056

Lab 3

Activity 1: Gaussian Elimination

Code:

```
1 import numpy as np
2
3 def f(A, b, x):
4     return np.dot(A, x) - b
5
6 def forward_elimination(A, b):
7     n = len(A)
8     for i in range(n):
9         max_row = max(range(i, n), key=lambda r: abs(A[r][i]))
10        if i != max_row:
11            A[[i, max_row]] = A[[max_row, i]]
12            b[[i, max_row]] = b[[max_row, i]]
13
14        for j in range(i+1, n):
15            factor = A[j][i] / A[i][i]
16            A[j] -= factor * A[i]
17            b[j] -= factor * b[i]
18    return A, b
19
20 def backward_substitution(A, b):
21     n = len(A)
22     x = np.zeros(n)
23     for i in range(n-1, -1, -1):
24         x[i] = (b[i] - np.dot(A[i, i+1:], x[i+1:])) / A[i, i]
25     return x
26
27 def gaussian_elimination(A, b):
28     A, b = forward_elimination(A, b)
29     return backward_substitution(A, b)
30
31 def solve_linear_system():
32     A = np.array([[3, 1, -2], [2, -2, 4], [-1, 12, -1]], dtype=float)
33     b = np.array([1, -2, 0], dtype=float)
34     solution = gaussian_elimination(A, b)
35     print(f"\nApproximate solution:\n x = {solution[0]}\n y = {solution[1]}\n z = {solution[2]}")
36
37 if __name__ == "__main__":
38     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 = -0.04347826086956521
z = -0.5217391304347826
```

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

Code:

```
1 import numpy as np
2
3 def jacobi_method(A, b, tol=1e-6, max_iter=100):
4     n = len(A)
5     x = np.zeros(n)
6     x_new = np.zeros(n)
7     iter_count = 0
8
9     print(f"{'Iter':<5}{'x':<15}{'y':<15}{'z':<15}")
10    print("-" * 60)
11
12    for _ in range(max_iter):
13        for i in range(n):
14            x_new[i] = (b[i] - np.sum(A[i, :i] * x[:i]) - np.sum(A[i, i+1:] * x[i+1:])) / A[i, i]
15
16        error = np.linalg.norm(x_new - x, ord=np.inf)
17        if iter_count < 3:
18            print(f"{'Iter':<5}{x_new[0]:<15.6f}{x_new[1]:<15.6f}{x_new[2]:<15.6f}")
19
20        if error < tol:
21            return x_new
22        x = x_new.copy()
23        iter_count += 1
24
25    return x_new
26
27 def gauss_seidel_method(A, b, tol=1e-6, max_iter=100):
28     n = len(A)
29     x = np.zeros(n)
30     iter_count = 0
31
32     print(f"{'Iter':<5}{'x':<15}{'y':<15}{'z':<15}")
33     print("-" * 60)
34
35     for _ in range(max_iter):
36         x_old = x.copy()
37         for i in range(n):
38             x[i] = (b[i] - np.sum(A[i, :i] * x[:i]) - np.sum(A[i, i+1:] * x[i+1:])) / A[i, i]
39
40         error = np.linalg.norm(x - x_old, ord=np.inf)
41         if iter_count < 3:
42             print(f"{'Iter':<5}{x[0]:<15.6f}{x[1]:<15.6f}{x[2]:<15.6f}")
43
44         if error < tol:
45             return x
46         iter_count += 1
47
48     return x
49
50 def solve_iterative_methods():
51     A = np.array([[5, -2, 3], [2, 5, -1], [1, 3, 5]], dtype=float)
52     b = np.array([10, 4, 8], dtype=float)
53
54     print("\nSolving using Jacobi Method:")
55     jacobi_solution = jacobi_method(A, b)
56     print(f"\nJacobi Solution in 100th approximation:\n x = {jacobi_solution[0]}\n y = {jacobi_solution[1]}\n z = {jacobi_solution[2]}")
57
58     print("\nSolving using Gauss-Seidel Method:")
59     gauss_seidel_solution = gauss_seidel_method(A, b)
60     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]}")
61
62 if __name__ == "__main__":
63     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:

```
1 import numpy as np
2 import time
3
4 def gaussian_elimination(A, b):
5     start_time = time.time()
6     A = A.astype(float)
7     b = b.astype(float)
8     n = len(A)
9
10    for i in range(n):
11        max_row = max(range(i, n), key=lambda r: abs(A[r][i]))
12        if i != max_row:
13            A[[i, max_row]] = A[[max_row, i]]
14            b[[i, max_row]] = b[[max_row, i]]
15
16        for j in range(i+1, n):
17            factor = A[j][i] / A[i][i]
18            A[j] -= factor * A[i]
19            b[j] -= factor * b[i]
20
21    x = np.zeros(n)
22    for i in range(n-1, -1, -1):
23        x[i] = (b[i] - np.dot(A[i, i+1:], x[i+1:])) / A[i, i]
24
25    end_time = time.time()
26    return x, end_time - start_time, 1
27
28 def jacobi_method(A, b, tol=1e-6, max_iter=100):
29     start_time = time.time()
30     n = len(A)
31     x = np.zeros(n)
32     x_new = np.zeros(n)
33     iter_count = 0
34
35     for _ in range(max_iter):
36         for i in range(n):
37             x_new[i] = (b[i] - np.sum(A[i, :i] * x[:i]) - np.sum(A[i, i+1:] * x[i+1:])) / A[i, i]
38
39         error = np.linalg.norm(x_new - x, ord=np.inf)
40         if error < tol:
41             end_time = time.time()
42             return x_new, end_time - start_time, iter_count
43         x = x_new.copy()
44         iter_count += 1
45
46     end_time = time.time()
47     return x_new, end_time - start_time, iter_count
48
49 def gauss_seidel_method(A, b, tol=1e-6, max_iter=100):
50     start_time = time.time()
51     n = len(A)
52     x = np.zeros(n)
53     iter_count = 0
54
55     for _ in range(max_iter):
56         x_old = x.copy()
57         for i in range(n):
58             x[i] = (b[i] - np.sum(A[i, :i] * x[:i]) - np.sum(A[i, i+1:] * x[i+1:])) / A[i, i]
59
60         error = np.linalg.norm(x - x_old, ord=np.inf)
61         if error < tol:
62             end_time = time.time()
63             return x, end_time - start_time, iter_count
64         iter_count += 1
65
66     end_time = time.time()
67     return x, end_time - start_time, iter_count
68
69 def solve_and_compare_methods():
70     A = np.array([[3, 1, -2], [2, -2, 4], [-1, 12, -1]], dtype=float)
71     b = np.array([1, -2, 0], dtype=float)
72
73     ge_solution, ge_time, ge_iters = gaussian_elimination(A.copy(), b.copy())
74     jacobi_solution, jacobi_time, jacobi_iters = jacobi_method(A.copy(), b.copy())
75     gs_solution, gs_time, gs_iters = gauss_seidel_method(A.copy(), b.copy())
76
77     print("\nComparison of Methods:\n")
78     print(f"Method: '{Method':<15}' x':<15}' y':<15}' z':<15}' Time (s)':<15}' Iterations':<10}")
79     print(f"{'-' * 80}")
80     print(f"Gaussian: '{ge_solution[0]:<15.6g}' {ge_solution[1]:<15.6g}' {ge_solution[2]:<15.6g}' {ge_time:<15.6f}' {'':<10}")
81     print(f"Jacobi: '{jacobi_solution[0]:<15.6g}' {jacobi_solution[1]:<15.6g}' {jacobi_solution[2]:<15.6g}' {jacobi_time:<15.6f}' {jacobi_iters:<10}")
82     print(f"Gauss-Seidel: '{gs_solution[0]:<15.6g}' {gs_solution[1]:<15.6g}' {gs_solution[2]:<15.6g}' {gs_time:<15.6f}' {gs_iters:<10}")
83
84 if __name__ == "__main__":
85     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:

```
1 import numpy as np
2
3 def jacobi_method(A, b, tol=1e-6, max_iter=100):
4     n = len(A)
5     x = np.zeros(n)
6     x_new = np.zeros(n)
7     iter_count = 0
8
9     print(f"\n{'Jacobi Method':^50}")
10    print(f"{'Iter':<5}{x':<15}{y':<15}{z':<15}")
11    print("-" * 50)
12
13    for _ in range(max_iter):
14        for i in range(n):
15            x_new[i] = (b[i] - np.sum(A[i, :i] * x[:i]) - np.sum(A[i, i+1:] * x[i+1:])) / A[i, i]
16
17            error = np.linalg.norm(x_new - x, ord=np.inf)
18            print(f"{'iter_count':<5}{x_new[0]:<15.6f}{x_new[1]:<15.6f}{x_new[2]:<15.6f}")
19
20            if error < tol:
21                return x_new, iter_count
22            x = x_new.copy()
23            iter_count += 1
24
25    return x_new, iter_count
26
27 def gauss_seidel_method(A, b, tol=1e-6, max_iter=100):
28     n = len(A)
29     x = np.zeros(n)
30     iter_count = 0
31
32     print(f"\n{'Gauss-Seidel Method':^50}")
33     print(f"{'Iter':<5}{x':<15}{y':<15}{z':<15}")
34     print("-" * 50)
35
36     for _ in range(max_iter):
37         x_old = x.copy()
38         for i in range(n):
39             x[i] = (b[i] - np.sum(A[i, :i] * x[:i]) - np.sum(A[i, i+1:] * x[i+1:])) / A[i, i]
40
41             error = np.linalg.norm(x - x_old, ord=np.inf)
42             print(f"{'iter_count':<5}{x[0]:<15.6f}{x[1]:<15.6f}{x[2]:<15.6f}")
43
44             if error < tol:
45                 return x, iter_count
46             iter_count += 1
47
48     return x, iter_count
49
50 def solve_iterative_methods():
51     A = np.array([[5, -2, 3], [2, 5, -1], [1, 3, 5]], dtype=float)
52     b = np.array([10, 4, 8], dtype=float)
53
54     jacobi_solution, jacobi_iters = jacobi_method(A, b)
55     gauss_seidel_solution, gauss_seidel_iters = gauss_seidel_method(A, b)
56
57     print("\nFinal Comparison:")
58     print(f"{'Method':<15}{Iterations':<10}")
59     print("-" * 30)
60     print(f"{'Jacobi':<15}{jacobi_iters:<10}")
61     print(f"{'Gauss-Seidel':<15}{gauss_seidel_iters:<10}")
62
63 if __name__ == "__main__":
64     solve_iterative_methods()
65
```

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
```