數值方法 作業 7

工科系 115　E14116401 張瑋哲

A 小題：

```
PS D:\ForClass\1132\1132Numerical\HW7> & C:/ProgramData/anaconda3/python.exe d:/ForClass/1132/1132Numerical/HW7/A.py

Jacobi method:
  @Times = 1: x = [ 0.   -0.25  2.25  1.    2.    1.5 ]
  @Times = 2: x = [0.1875 0.8125 2.0625 1.875  2.5625 2.5625]
  @Times = 3: x = [0.671875 0.953125 2.453125 2.328125 3.3125   2.65625 ]
  @Times = 4: x = [0.8203125  1.359375   2.32421875 2.66015625 3.484375   2.94140625]
  @Times = 5: x = [1.00488281 1.40722656 2.45410156 2.81152344 3.74023438 2.95214844]
  @Times = 6: x = [1.0546875  1.54980469 2.40478516 2.92431641 3.79272461 3.04858398]
  @Times = 7: x = [1.11853027 1.56304932 2.45141602 2.97399902 3.88067627 3.04937744]
  @Times = 8: x = [1.13426208 1.61265564 2.43293762 3.012146   3.89660645 3.08302307]
  @Times = 9: x = [1.15620041 1.61595154 2.44976807 3.0284729  3.92695618 3.08238602]
  @Times = 10: x = [1.16110611 1.63323116 2.44284534 3.04138565 3.93170261 3.09418106]
  @Times = 11: x = [1.1686542  1.63391352 2.4489274  3.04674745 3.94219947 3.09363699]
  @Times = 12: x = [1.17016524 1.63994527 2.44633776 3.05112267 3.94357449 3.09778172]
  @Times = 13: x = [1.17276698 1.64001937 2.44853812 3.05288036 3.94721241 3.09747806]
  @Times = 14: x = [1.17322493 1.64212938 2.44757126 3.05436436 3.94759445 3.09893763]
  @Times = 15: x = [1.17412344 1.64209766 2.44836814 3.05493925 3.94885784 3.09879143]
  @Times = 16: x = [1.17425923 1.64283736 2.44800781 3.05544318 3.94895708 3.0993065 ]
  @Times = 17: x = [1.17457013 1.64280603 2.44829669 3.0556307  3.94939676 3.09924122]
  @Times = 18: x = [1.17460918 1.6430659  2.44816262 3.05580203 3.94941949 3.09942336]
  @Times = 19: x = [1.17471698 1.64304782 2.44826744 3.05586301 3.94957282 3.09939553]
  @Times = 20: x = [1.17472771 1.64313931 2.44821763 3.05592133 3.94957659 3.09946007]
  @Times = 21: x = [1.17476516 1.64313048 2.4482557  3.05594109 3.94963018 3.09944856]
  @Times = 22: x = [1.17476789 1.64316276 2.44823722 3.05596097 3.94963003 3.09947147]
  @Times = 23: x = [1.17478093 1.64315879 2.44825105 3.05596735 3.9496488  3.09946681]
  @Times = 24: x = [1.17478153 1.6431702  2.4482442  3.05597414 3.94964824 3.09947496]
  @Times = 25: x = [1.17478608 1.64316849 2.44824923 3.05597618 3.94965482 3.09947311]
  @Times = 26: x = [1.17478617 1.64317253 2.44824669 3.0559785  3.94965445 3.09947601]
  @Times = 27: x = [1.17478776 1.64317183 2.44824853 3.05597916 3.94965676 3.09947529]
  @Times = 28: x = [1.17478775 1.64317326 2.44824759 3.05597995 3.94965657 3.09947632]
  @Times = 29: x = [1.1747883  1.64317298 2.44824825 3.05598016 3.94965738 3.09947604]
  @Times = 30: x = [1.17478828 1.64317349 2.44824791 3.05598043 3.94965729 3.09947641]
  @Times = 31: x = [1.17478848 1.64317337 2.44824815 3.0559805  3.94965758 3.0994763 ]
  @Times = 32: x = [1.17478847 1.64317355 2.44824802 3.05598059 3.94965754 3.09947643]
  @Times = 33: x = [1.17478854 1.64317351 2.44824811 3.05598061 3.94965764 3.09947639]
  @Times = 34: x = [1.17478853 1.64317357 2.44824806 3.05598064 3.94965763 3.09947644]
  @Times = 35: x = [1.17478855 1.64317356 2.4482481  3.05598065 3.94965766 3.09947642]
  @Times = 36: x = [1.17478855 1.64317358 2.44824808 3.05598066 3.94965766 3.09947644]
  @Times = 37: x = [1.17478856 1.64317357 2.44824809 3.05598066 3.94965767 3.09947643]
  converge in 38 iterations.
Solution by jacobi method: [1.17479 1.64317 2.44825 3.05598 3.94966 3.09948]
```

圖 1、A 小題(Jacobi method)計算結果

```python
import numpy as np
A = np.array([
    [4, -1,  0, -1,  0,  0],
    [-1, 4, -1,  0, -1,  0],
    [0, -1,  4,  0,  1, -1],
    [-1, 0,  0,  4, -1, -1],
    [0, -1,  0, -1,  4, -1],
    [0,  0, -1,  0, -1,  4]
], dtype=float)

b = np.array([0, -1, 9, 4, 8, 6], dtype=float)

# 猜測值設定
x_initial = np.zeros_like(b)
# 收斂條件
max_iterations = 1000 #上限次數
tolerance = 1e-8 #小數上限

def jacobi_method(A, b, x_initial, tol=1e-8, max_iter=1000):
    n = len(b)
    x = x_initial.copy()
    x_new = np.zeros_like(x)

    print("\nJacobi method:")
    for k in range(max_iter):
        for i in range(n):
            sigma = 0
            for j in range(n):
                if i != j:
                    sigma += A[i, j] * x[j]
            x_new[i] = (b[i] - sigma) / A[i, i]

        if np.linalg.norm(x_new - x, np.inf) < tol:
            print(f"  converge in {k+1} iterations.") #在幾次內收斂
            return x_new
        x = x_new.copy()  #每一次的計算結果
        print(f"  @Times={k+1}: x = {x}")

    print("  Maybe not convergence.")
    return x

x_jacobi = jacobi_method(A, b, x_initial.copy(), tolerance, max_iterations)
x_jacobi = np.round(x_jacobi, 5)
print(f"Solution by jacobi method: {x_jacobi}")
```

圖 2、A 小題(Jacobi method)程式

B 小題：

```
PS D:\ForClass\1132\1132Numerical\HW7> & C:/ProgramData/anaconda3/python.exe d:/ForClass/1132/1132Numerical/HW7/B.py

Gauss-Seidel :
  Iteration 1: [ 0.        -0.25       2.1875     1.         2.1875     2.59375]
  Iteration 2: [0.1875     0.890625   2.57421875 2.2421875  3.43164062 3.00146484]
  Iteration 3: [0.78320312 1.44726562 2.50427246 2.80407715 3.8132019  3.07936859]
  Iteration 4: [1.06283569 1.59507751 2.46531105 2.98885155 3.91582441 3.09528387]
  Iteration 5: [1.14598227 1.63177943 2.45280972 3.03927264 3.94158398 3.09859843]
  Iteration 6: [1.16776302 1.64053918 2.44938841 3.05198636 3.94778099 3.09929235]
  Iteration 7: [1.17313138 1.6425752  2.44852164 3.05505118 3.94922968 3.09943783]
  Iteration 8: [1.17440659 1.64303948 2.44831191 3.05576853 3.94956146 3.09946834]
  Iteration 9: [1.174702   1.64314384 2.44826268 3.05593295 3.94963628 3.09947474]
  Iteration 10: [1.1747692  1.64316704 2.44825137 3.05597006 3.94965296 3.09947608]
  Iteration 11: [1.17478427 1.64317215 2.44824882 3.05597833 3.94965664 3.09947637]
  Iteration 12: [1.17478762 1.64317327 2.44824825 3.05598016 3.94965745 3.09947642]
  Iteration 13: [1.17478836 1.64317351 2.44824812 3.05598056 3.94965762 3.09947644]
  Iteration 14: [1.17478852 1.64317357 2.44824809 3.05598064 3.94965766 3.09947644]
  Iteration 15: [1.17478855 1.64317358 2.44824809 3.05598066 3.94965767 3.09947644]
  Iteration 16: [1.17478856 1.64317358 2.44824809 3.05598067 3.94965767 3.09947644]
  converge in 16 iterations.
Solution by Gauss-Seidel: [1.17479 1.64317 2.44825 3.05598 3.94966 3.09948]
```

圖 3、B 小題(Gauss-seidel)計算結果

```python
import numpy as np
A = np.array([
    [4, -1,  0, -1,  0,  0],
    [-1, 4, -1,  0, -1,  0],
    [0, -1,  4,  0,  1, -1],
    [-1, 0,  0,  4, -1, -1],
    [0, -1,  0, -1,  4, -1],
    [0,  0, -1,  0, -1,  4]
], dtype=float)

b = np.array([0, -1, 9, 4, 8, 6], dtype=float)

# 猜測值設定
x_initial = np.zeros_like(b)
# 收斂條件
max_iterations = 1000 #上限次數
tolerance = 1e-8     #小數上限

def gauss_seidel_method(A, b, x_initial, tol=1e-8, max_iter=1000):
    n = len(b)
    x = x_initial.copy()

    print("\nGauss-Seidel :")
    for k in range(max_iter):
        x_old = x.copy()
        for i in range(n):
            sigma1 = 0
            for j in range(i): # j < i
                sigma1 += A[i, j] * x[j] # 使用本輪已更新的 x[j]
            sigma2 = 0
            for j in range(i + 1, n): # j > i
                sigma2 += A[i, j] * x_old[j] # 使用上一輪的 x_old[j]
            x[i] = (b[i] - sigma1 - sigma2) / A[i, i]
        print(f"  @Times= {k+1}: {x}") #每一次的計算結果
        if np.linalg.norm(x - x_old, np.inf) < tol:
            print(f"  converge in {k+1} iterations.") #在幾次內收斂
            return x

    print("  Maybe not convergence.")
    return x

x_gauss_seidel = gauss_seidel_method(A, b, x_initial.copy(), tolerance, max_iterations)
x_gauss_seidel = np.round(x_gauss_seidel, 5)
print(f"Solution by Gauss-Seidel: {x_gauss_seidel}")
```

圖 4、B 小題(Gauss-seidel)程式碼

C 小題：

```
PS D:\ForClass\1132\1132Numerical\HW7> & C:/ProgramData/anaconda3/python.exe d:/ForClass/1132/1132Numerical/HW7/C.py

SOR method (omega = 1.1):
  Iteration 1: [ 0.         -0.275       2.399375    1.1         2.426875    2.97721875]
  Iteration 2: [0.226875   1.14210937 2.70048711 2.53851641 3.78821975 3.13667251]
  Iteration 3: [0.98948459 1.66729171 2.48428102 3.02260199 3.97348373 3.11221806]
  Iteration 4: [1.19077231 1.66161852 2.44666893 3.07377018 3.96074348 3.10081661]
  Iteration 5: [1.18315466 1.64624409 2.44657034 3.05991954 3.95084522 3.09920762]
  Iteration 6: [1.17587953 1.64303174 2.44797635 3.05613945 3.94946965 3.09937689]
  Iteration 7: [1.17468412 1.64303261 2.44826082 3.05585699 3.94957632 3.09946753]
  Iteration 8: [1.17472623 1.64315167 2.44826071 3.05595107 3.94964919 3.09947847]
  Iteration 9: [1.17478063 1.64317473 2.44825003 3.05597967 3.94965912 3.09947717]
  Iteration 10: [1.1747894  1.64317463 2.44824798 3.0559816  3.94965827 3.0994765 ]
  Iteration 11: [1.17478902 1.64317374 2.44824799 3.05598088 3.94965773 3.09947642]
  Iteration 12: [1.17478862 1.64317357 2.44824807 3.05598067 3.94965766 3.09947643]
  Iteration 13: [1.17478856 1.64317357 2.44824809 3.05598066 3.94965767 3.09947644]
  converge in 14 iterations.
solution by SOR method (omega=1.1): [1.17479 1.64317 2.44825 3.05598 3.94966 3.09948]
```

圖 5 、C 小題(SOR method)計算結果

```python
import numpy as np
A = np.array([
    [4, -1,  0, -1,  0,  0],
    [-1, 4, -1,  0, -1,  0],
    [0, -1,  4,  0,  1, -1],
    [-1, 0,  0,  4, -1, -1],
    [0, -1,  0, -1,  4, -1],
    [0,  0, -1,  0, -1,  4]
], dtype=float)

b = np.array([0, -1, 9, 4, 8, 6], dtype=float)

# 猜測值設定
x_initial = np.zeros_like(b)
# 收斂條件
max_iterations = 1000 #上限次數
tolerance = 1e-8 #error範圍

def sor_method(A, b, x_initial, omega, tol=1e-8, max_iter=1000):
    n = len(b)
    x = x_initial.copy()


    print(f"\nSOR method (omega = {omega}):")
    for k in range(max_iter):
        x_old = x.copy()
        for i in range(n):
            sigma1 = 0
            for j in range(i): # j < i
                sigma1 += A[i, j] * x[j] # 使用本輪已更新的 x[j]
            sigma2 = 0
            for j in range(i + 1, n): # j > i
                sigma2 += A[i, j] * x_old[j] # 使用上一輪的 x_old[j]

            x_gauss_seidel_i = (b[i] - sigma1 - sigma2) / A[i, i]
            x[i] = (1 - omega) * x_old[i] + omega * x_gauss_seidel_i

        if np.linalg.norm(x - x_old, np.inf) < tol:
            print(f"  converge in {k+1} iterations.") #在幾次內收斂
            return x

        print(f"  Iteration {k+1}: {x}") #每一次的計算結果
```

```python
    print("  Maybe not convergence.")
    return x

omega_sor = 1.1
x_sor = sor_method(A, b, x_initial.copy(), omega_sor, tolerance, max_iterations)
x_sor = np.round(x_sor, 5)
print(f"solution by SOR method (omega={omega_sor}): {x_sor}")
```

圖 6、C 小題(SOR method)程式碼

D 小題：

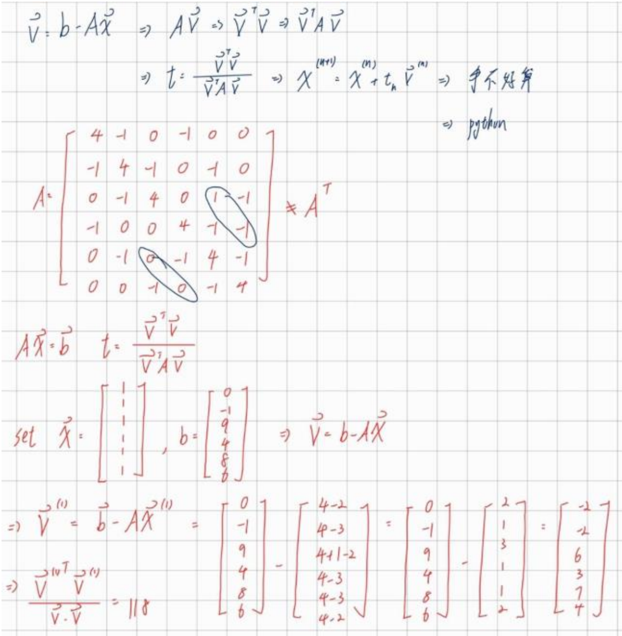

圖 7、D 小題(Conjugate Gradient Method)部分手算及過程描述

```
PS D:\ForClass\1132\1132Numerical\HW7> & C:/ProgramData/anaconda3/python.exe d:/ForClass/1132/1132Numerical/HW7/D.py

最終解:
x[1] = 1.17479
x[2] = 1.64317
x[3] = 2.44825
x[4] = 3.05598
x[5] = 3.94966
x[6] = 3.09948
```

圖 8、D 小題(Conjugate Gradient Method)計算結果

```python
import numpy as np

def conjugate_gradient(A, b, x_init, tol=1e-8, max_iter=1000):

    x = np.array(x_init, dtype=float)
    A = np.array(A, dtype=float)
    b = np.array(b, dtype=float)

    for k in range(max_iter):

        v = b - (A @ x)
        Ar = A @ v
        rTr = v @ v
        rTAr = v @ Ar

        alpha = rTr / rTAr
        # 更新解向量 x(n+1) = x(n) + alpha * v(n)

        x_new = x + alpha * v
        # 計算誤差
        error = np.linalg.norm(x_new - x)
        x = x_new
        #更新x

        if error < tol:
            break
    else: # not converge
        print(f"\nMaybe not convergence.")

    return x
```

```python
if __name__ == "__main__":
    n = 6

    # 初始化矩陣 A
    A_matrix = np.array([
    [4, -1,  0, -1,  0,  0],
    [-1, 4, -1,  0, -1,  0],
    [0, -1,  4,  0,  1, -1],
    [-1, 0,  0,  4, -1, -1],
    [0, -1,  0, -1,  4, -1],
    [0,  0, -1,  0, -1,  4]
    ], dtype=float)

    b_vector = np.array([0, -1, 9, 4, 8, 6], dtype=float)

    # 猜測值設定
    x_initial = np.zeros(n)

    # 收斂條件
    max_iterations = 1000 #上限次數
    tolerance = 1e-8 #小數上限

    solution_x = conjugate_gradient(A_matrix, b_vector, x_initial,
                                    tol=tolerance, max_iter=max_iterations)

    # 輸出結果
    print("\n最終解:")
    for i in range(n):
        print(f"x[{i + 1}] = {solution_x[i]:.5f}")
```

圖 9、D 小題(Conjugate Gradient Method)程式碼