A5-Q3: Jacobi and Gauss-Seidel

Prelims

In [1]:
```
import numpy as np
from scipy.linalg import solve_triangular
import matplotlib.pyplot as plt
```

In [2]:
```
def split_diag(A):
    D = np.zeros_like(A)
    L_plus_U = A.copy()

    for i in range(len(A)):
        for j in range(len(A)):
            if i == j:
                D[i, j] = L_plus_U[i, j]
                L_plus_U[i, j] = 0

    return D, L_plus_U

def decomposition(A):

    D, L_plus_U = split_diag(A)

    U_indices = np.triu_indices(n = A.shape[0], m = A.shape[1])
    U = np.zeros_like(A)

    L = L_plus_U.copy()
    L[U_indices] = U[U_indices]
    U[U_indices] = L_plus_U[U_indices]

    return D, L, U
```

(a): `Jacobi_step`

In [3]:
```
def Jacobi_step(A, f, u0):
    '''
     u = Jacobi_step(A, f, u0)
     Performs one Jacobi iteration on
     A u = f using the initial guess of u0.
    '''

    D, L, U = decomposition(A)

    results = np.zeros_like(u0, dtype = 'float64')
    unscaled_product = (f - (np.dot((L + U), u0)))

    for i in range(D.shape[0]):
        results[i] = unscaled_product[i] / D[i, i]

    return results
```

(b): `GS_step`

In [4]:
```
def GS_step(A, f, u0):
    '''
     u = GS_step(A, f, u0)
     Performs one Gauss-Seidel iteration on
     A u = f using the initial guess of u0.
    '''

    D, L, U = decomposition(A)
    return solve_triangular((L + D), f - (np.dot(U, u0)), lower = True)
```

(c): `iterative_solve`

In [13]:
```
def iterative_solve(StepFcn, A, f, u0, maxIters=50, tol=1.e-5):
    '''
     u = iterative_solve(StepFcn, A, f, u0, maxIters=50, tol=1e-5)

     Iteratively solves A u = f, taking steps specified by the
     function StepFcn.
```

```
    Inputs:
     StepFcn  function that takes a single step. Its calling signature
              must be
                u = StepFcn(A, f, u0)
     A        (N,N) array holding the system matrix
     f        (N,) vector, RHS vector of the system
     u0       (N,) vector, initial guess at the solution
     maxIters maximum number of iterations
     tol      tolerance for stopping criterion. Convergence is considered
              successful if the 2-norm of the residual is less than tol.

              If the maxIter is reached and the tolerance is still not
              satisfied, then the function should print a message stating
              that the method did not successfully converge.

    Outputs:
     u        (N,) vector approximation of the solution
    '''
    N = len(f)
    u = u0.copy()
    r = np.ones_like(f)
    count = 0

    while (np.linalg.norm(r) > tol):

        v = StepFcn(A, f, u)
        r = f - (np.dot(A, v))

        count += 1
        u = v.copy()

        # If any of the elements of r to NaN, then it has blown up to cause an overflow
        if (np.isnan(r).any() or count > maxIters):
            print("Fails to converge")
            return None

    return v
```

In [6]:

```
def diagonally_dominant(A):

    diags = np.zeros(A.shape[0])
    off_diags = np.zeros(A.shape[0])

    for i in range(A.shape[0]):
        off_diagonal_sum = 0
        diag = A[i, i]
        for j in range(A.shape[1]):
            if i != j:
                off_diagonal_sum += abs(A[i, j])

        if off_diagonal_sum > diag:
            return False, (i, off_diagonal_sum, diag)

    return True, _
```

(d)

In [7]:

```
def will_converge(A):
    diag_dominant, _ = diagonally_dominant(A)
    if diag_dominant:
        return "Matrix is diagonally dominant, can use J or GS, both are guaranteed to converge"

    pd = np.all(np.linalg.eigvals(A) > 0)
    sym = np.allclose(A, A.T, atol=1e-12)

    if pd and sym:
        return "Matrix is positive definite and symmetric, but not diagonally dominant. Gauss-Seidel will conve

    return "May or may not converge"
```

System 1

In [8]:

```
# Load the system from the file system1.npz
data1 = np.load('system1.npz')
A1 = data1['A']
f1 = data1['f']
```

In [9]:

```
# Determine if any of the convergence theorems apply.
v1_J = iterative_solve(Jacobi_step, A1, f1, np.ones_like(f1), maxIters = 1000, tol = 10e-4)
v1_GS= iterative_solve(GS_step, A1, f1, np.ones_like(f1), maxIters = 1000, tol = 10e-4)

print("A1 " + will_converge(A1))
```

A1 Matrix is diagonally dominant, can use J or GS, both are guaranteed to converge

In [14]:
```
#np.zeros_like(f1)
print("Jacobi Residual Solution: ")
print(v1_J)
print()
print("Gauss-Seidel Solution: ")
print(v1_GS)
```

```
Jacobi Residual Solution:
[ 0.18626077 -0.29574517  0.18334608  0.24363594  0.10050612  0.11484692
 -0.24841585  0.13456753 -0.18453202 -0.01292882  0.12313881  0.10982869
  0.30751189  0.06963749 -0.01483954 -0.31279143 -0.28498091  0.16762298
 -0.12399915  0.16439319 -0.05634716 -0.12282899  0.0074121   0.32324275
  0.1360699  -0.25812647 -0.16149191  0.25964131 -0.30784832 -0.31384631
 -0.01107099  0.27448764 -0.06647953 -0.30068649  0.02742924  0.13359014
 -0.29013877  0.07675133 -0.18676857  0.08094235 -0.22579644 -0.23352478
 -0.00375997 -0.17073687  0.2877989  -0.26416977  0.3187099  -0.25343944
 -0.07887928  0.19313376 -0.06770649 -0.16295874 -0.25992555  0.23410182
  0.2675301  -0.24503237  0.14884089 -0.0106379   0.1294775   0.32061091
  0.00957128  0.30499456  0.32135518  0.17668147 -0.23259707 -0.13461717
 -0.08865314  0.13906174  0.19389769 -0.23647357 -0.24721324  0.1518979
 -0.25950526 -0.2867984  -0.20071661 -0.05960143  0.16070971 -0.00880309
  0.14317258  0.23402334 -0.04521983  0.0422134  -0.0857719  -0.15669817
 -0.04108675  0.2479919  -0.31298434 -0.1653307   0.02340361  0.16317899
  0.05672658  0.06732578  0.30371574 -0.13163296  0.19054328  0.03121067
  0.14823578  0.22117674  0.01918888 -0.23980464]

Gauss-Seidel Solution:
[ 0.18626076 -0.29574523  0.1833447   0.24363625  0.10050564  0.11484667
 -0.24841748  0.13456873 -0.18453127 -0.01292812  0.12313865  0.10982829
  0.30751226  0.06963823 -0.01483958 -0.31279271 -0.28498278  0.16762201
 -0.12399695  0.16439394 -0.05634629 -0.1228292   0.00741185  0.32324268
  0.13607113 -0.25812619 -0.16149246  0.25963929 -0.3078479  -0.31384691
 -0.01107182  0.27448858 -0.06647885 -0.3006874   0.02742857  0.13359002
 -0.29013864  0.07675068 -0.18676937  0.08094323 -0.22579589 -0.2335252
 -0.00375959 -0.1707374   0.28779797 -0.26416965  0.31871014 -0.25343939
 -0.07887838  0.19313355 -0.06770755 -0.16295831 -0.25992576  0.23410209
  0.26752951 -0.24503316  0.14883918 -0.01063783  0.12947716  0.32061224
  0.00956992  0.30499398  0.32135384  0.17668014 -0.23259639 -0.13461818
 -0.08865443  0.13906101  0.19389742 -0.23647335 -0.24721343  0.15189763
 -0.25950617 -0.28679822 -0.20071657 -0.05960123  0.16070884 -0.0088024
  0.14317273  0.23402364 -0.04522008  0.04221354 -0.08577102 -0.15669756
 -0.04108755  0.24799269 -0.3129845  -0.16533065  0.02340489  0.16317912
  0.05672558  0.06732573  0.30371606 -0.13163147  0.1905425   0.03121014
  0.14823577  0.22117642  0.01918812 -0.23980361]
```

System 2

In [11]:
```
# Load the system from the file system2.npz
data2 = np.load('system2.npz')
A2 = data2['A']
f2 = data2['f']

print("A2 " + will_converge(A2))
```

A2 Matrix is positive definite and symmetric, but not diagonally dominant. Gauss-Seidel will converge, Jacobi may or may not

In [12]:
```
# Determine if any of the convergence theorems apply.

print("Jacobi Solution: ")
v2_J = iterative_solve(Jacobi_step, A2, f2, np.ones_like(f2), maxIters = 1000, tol = 10e-4)

print()
print("Gauss-Seidel Solution: ")
v2_GS = iterative_solve(GS_step, A2, f2, np.ones_like(f2), maxIters = 1000, tol = 10e-4)
print(v2_GS)
```

```
Jacobi Solution:
Fails to converge

Gauss-Seidel Solution:
[  0.1731696   10.98640521  23.6189771    1.95344247 -43.65139767
   5.59811439  16.45570036  35.2086366   38.75276783   3.50752706
  22.52374877  13.92071161 -10.93794218 -27.68185818  25.0568833
  16.33403932 -28.86864363   1.48727979  -0.8657803   10.21570819
  23.97059917   3.61065372  26.97137878  12.67527954   7.40420886
  21.94632824  -2.62305615 -11.57373843   7.07059185  -4.15880915
 -28.8711918  -12.30607452  12.10580992 -22.05571482  15.88597387
 -14.28549582  15.80777293 -14.44355126   6.79712468  34.20076013
  23.90330789  -0.40884063  -8.01393852  19.52147715  28.47746912
 -15.26490604   8.06301365  14.07748777  -3.37276567 -23.20402621
  19.58102246 -14.92904276  22.8077017  -37.48150127  16.5001963
  -0.62921321   7.74244401  -5.18591774   3.25575792   3.41043358
   6.42927558   1.20309054  -1.77919057  15.48259687   7.24355538
  -2.07956949   7.32327812  -6.47574818  26.01237698  -6.02358927
  -1.65911132  -7.56743388  35.70696881   2.38929952 -17.00985487
   1.20833237  38.31626778  11.47289511   7.4478205    4.11094251
   7.54483876   8.99096718 -14.2171024   13.8770118   -3.40327075
  -5.75098524  19.8479811   -4.23591779  -3.68414131  -4.54104244
  21.74834637  -9.58283816   7.14588306  -8.3222187   -7.2482662
  41.01734001 -15.92469823  12.82502165 -25.01596066  26.31608077
  -7.68124244 -14.21142907 -17.86541181 -13.42442472   5.28395235
   6.07990554  19.94118316  34.18107509  19.69906771 -25.49018539]
```

The Jacobi method does not converge in this example. This is because A2 is not diagonally dominant it cannot be guaranteed to converge. However even in the non diagonally dominant case, the Gauss-Seidel method still converges to a solution, this is because the matrix A2 is positive definite and symmetric. As I increase the maxIters parameter, the norm of the residual of the GS methods goes to 0, while the norm of the residual for the Jacobi method goes to inf.