ORTA DOĞU TEKNİK ÜNİVERSİTESİ
MIDDLE EAST TECHNICAL UNIVERSITY

MATH 381 BONUS CODING HOMEWORK REPORT

GROUP MEMBERS:

Ali Valiyev (2415461) – 15/07/2002

Orkhan Ashrafov (2418408) – 30/08/2002

QUESTION 1:

Codes to solve a system of n linear equations and n unknowns using:

a) Jacobi iterative method

QUESTION 1

```
[28] # JACOBI method

     import numpy as np

     def Jacobi(n, M, x):
       A = M[:, :-1]  # Extracting coefficient matrix A from augmented matrix M
       b = M[:, -1] # Extracting right hand side vector from augmented matrix M
       D = np.diag(A)  # extract diagonal elements of A
       R = A - np.diagflat(D)  # R = A - D
       print("k\tx(k)\tRele")  # print column headers
       k=1
       rele=1000000 # We create dummy rele variable and set it practically high enough so that we're able to execute first iteration, then we update rele accordingly after first iteration
       while rele>=10**(-3): # Let's use tolerance=0.001 as given in QUESTION 2
         if k>=11:
           # we stop here because we are required to print only first 10 iterations as output. Since we begin from k=0, k=10 would be 11th iteration, so we stop.
           break
         x_new = (b - np.dot(R, x)) / D # solution of Jacobi method
         rele = np.linalg.norm(x_new - x, ord=np.inf) / np.linalg.norm(x_new, ord=np.inf)  # calculate relative error
         print("{}\t{}\t{:.6f}".format(k, x_new, rele))  # print iteration number, solution, and relative error
         x = x_new
         k=k+1
       return x
```

b) Gauss-Seidel method

```python
# GAUSS-SEIDEL method

import numpy as np
def Gauss_Seidel(n, M, x):
  A = M[:, :-1] # Extracting coefficient matrix A from augmented matrix M
  b = M[:, -1]  # Extracting right hand side vector from augmented matrix M
  L = np.tril(A) # Extract lower triangular part elements of A
  U = A - L
  print("k\tx(k)\tRele")  # print column headers
  k=1
  rele=1000000 # We create dummy rele variable and set it practically high enough so that we're able to execute first iteration, then we update rele accordingly after first iteratio
  while rele>=10**(-3): # Let's use tolerance=0.001 as given in QUESTION 2
    if k>=11:
      # we stop here because we are required to print only first 10 iterations as output. Since we begin from k=0, k=10 would be 11th iteration, so we stop.
      break
    x_neww = np.dot(np.linalg.inv(L), b - np.dot(U, x))
    rele = np.linalg.norm(x_neww - x, ord=np.inf) / np.linalg.norm(x_neww, ord=np.inf)
    print("{}\t{}\t{:.6f}".format(k, x_neww, rele))  # print iteration number, solution, and relative error
    x=x_neww
    k=k+1
  return x_neww
```

QUESTION 1:

Results of tests our codes:

a) Jacobi method

```
# TESTING OUR RESULTS with Jacobi for the linear system given in QUESTION 1
M = np.array([[2., -1., 2., -1., -1],
              [2., 1., -2., -2., -2],
              [-1., 2., -4., 1., 1],
              [3.0, 0., 0., -3., -3]])  # Augmented matrix
x0 = np.array([1., 1., 1., 1.])  #LET'S START WITH STARTING VECTOR [1, 1, 1, 1]^T
x = Jacobi(4, M, x0);
print(x)
```

```
k       x(k)    Rele
1       [-0.5   0.    0.25  2.  ]       0.750000
2       [0.25  3.5   0.375 0.5  ]       1.000000
3       [ 1.125  -0.75    1.5625  1.25  ]       2.720000
4       [-1.8125   1.375   -0.59375  2.125  ]   1.382353
5       [ 1.84375   4.6875    1.421875 -0.8125  ]       0.780000
6       [ 0.015625  -4.46875    1.4296875  2.84375  ]   2.048951
7       [-2.7421875   6.515625   -1.77734375  1.015625  ]       1.685851
8       [ 5.04296875  1.9609375   3.94726562 -1.7421875 ]       1.543765
9       [-4.33789062 -7.67578125 -0.96582031  6.04296875]       1.255471
10      [-0.35058594 16.83007812 -1.49267578 -3.33789062]       1.456075
```

b) Gauss-Seidel method

```
[31] # TESTING OUR RESULTS with Gauss-Seidel for the linear system given in QUESTION 1
     M = np.array([[2., -1., 2., -1., -1],
                   [2., 1., -2., -2., -2],
                   [-1., 2., -4., 1., 1],
                   [3.0, 0., 0., -3., -3]]) # Augmented matrix
     x0 = np.array([1., 1., 1., 1.]) # LET'S START WITH STARTING VECTOR [1, 1, 1, 1]^T
     x = Gauss_Seidel(4, M, x0);
     print(x)
```

```
k       x(k)    Rele
1       [-0.5    3.     1.625  0.5  ]   0.666667
2       [-0.375    3.      1.46875  0.625  ]   0.052083
3       [-0.15625    2.5       1.1953125  0.84375  ]   0.200000
4       [-0.0234375   2.125      1.02929688  0.9765625 ]      0.176471
5       [0.02148438 1.96875    0.97314453 1.02148438]   0.079365
6       [0.02197266 1.9453125  0.97253418 1.02197266]   0.012048
7       [0.0111084  1.96679688 0.9861145  1.0111084 ]   0.010924
8       [0.00283813 1.98876953 0.99645233 1.00283813]   0.011048
9       [-6.48498535e-04  1.99987793e+00  1.00081062e+00  9.99351501e-01]      0.005555
10      [-1.19590759e-03  2.00271606e+00  1.00149488e+00  9.98804092e-01]      0.001417
```

Tuesday, January 3, 2023 7:43 AM

QUESTION 2:

In this example, 10 iterations are not enough to see whether Jacobi method converges. So, we use exact same code as in **QUESTION 1**, but this time with **50** iterations.

a) Jacobi method

QUESTION 2

```python
# Jacobi method

import numpy as np

def Jacobi(n, M, x):
  A = M[:, :-1]  # Extracting coefficient matrix A from augmented matrix M
  b = M[:, -1] # Extracting right hand side vector from augmented matrix M
  D = np.diag(A)  # extract diagonal elements of A
  R = A - np.diagflat(D)  # R = A - D
  print("k\tx(k)\tRele")  # print column headers
  k=1
  rele=1000000 # We create dummy rele variable and set it practically high enough so that we're able to execute first iteration, then we update rele accordingly after first iteration
  while rele>=10**(-3): # Let's use tolerance=0.001 as given in QUESTION 2
    if k>=51: # printing only 50 iterations
      break
    x_new = (b - np.dot(R, x)) / D
    rele = np.linalg.norm(x_new - x, ord=np.inf) / np.linalg.norm(x_new, ord=np.inf)  # calculate relative error
    print("{}\t{}\t{:.6f}".format(k, x_new, rele))  # print iteration number, solution, and relative error
    x = x_new
    k=k+1
  return x
```

b) Gauss-Seidel method

[33]

```
# GAUSS-SEIDEL method


import numpy as np
def Gauss_Seidel(n, M, x):
  A = M[:, :-1] # Extracting coefficient matrix A from augmented matrix M
  b = M[:, -1]  # Extracting right hand side vector from augmented matrix M
  L = np.tril(A) # Extract lower triangular part elements of A
  U = A - L
  print("k\tx(k)\tRele")  # print column headers
  k=1
  rele=1000000 # We create dummy rele variable and set it practically high enough so that we're able to execute first iteration, then we update rele accordingly after first iteration
  while rele>=10**(-3): # Let's use tolerance=0.001 as given in QUESTION 2
    if k>=51:
      # printing only 50 iterations
      break
    x_neww = np.dot(np.linalg.inv(L), b - np.dot(U, x))
    rele = np.linalg.norm(x_neww - x, ord=np.inf) / np.linalg.norm(x_neww, ord=np.inf)
    print("{}\t{}\t{:.6f}".format(k, x_neww, rele))  # print iteration number, solution, and relative error
    x=x_neww
    k=k+1
  return x_neww
```

QUESTION 2:

Testing results of codes for given linear system

a) Jacobi method
So, we see results below that Jacobi method **converges** to solution [1,1,1]^T, which is indeed solution of the given linear system of equations.

```
k         x(k)    Rele
1         [ 2. -0. -0.]   1.000000
2         [2.          2.          0.66666667]      1.000000
3         [1.33333333 2.          2.         ]      0.666667
4         [0.          1.33333333 1.77777778]      0.750000
5         [ 0.22222222 -0.          0.88888889]     1.500000
6         [1.11111111 0.22222222 0.07407407]      0.800000
7         [1.92592593 1.11111111 0.51851852]      0.461538
8         [1.48148148 1.92592593 1.38271605]      0.448718
9         [0.61728395 1.48148148 1.77777778]      0.486111
10        [0.22222222 0.61728395 1.19341564]      0.724138
11        [0.80658436 0.22222222 0.48559671]      0.877551
12        [1.51440329 0.80658436 0.4170096 ]      0.467391
13        [1.5829904  1.51440329 1.04252401]      0.447140
14        [0.95747599 1.5829904  1.53726566]      0.395147
15        [0.46273434 0.95747599 1.3744856 ]      0.455090
16        [0.6255144  0.46273434 0.79256211]      0.734231
17        [1.20743789 0.6255144  0.51699436]      0.481949
18        [1.48300564 1.20743789 0.8194889 ]      0.392395
19        [1.1805111  1.48300564 1.29929381]      0.323535
20        [0.70070619 1.1805111  1.38217413]      0.347138
21        [0.61782587 0.70070619 1.02057613]      0.470131
22        [0.97942387 0.61782587 0.67307942]      0.369195
23        [1.32692058 0.97942387 0.73835854]      0.272509
24        [1.26164146 1.32692058 1.09525611]      0.268967
25        [0.90474389 1.26164146 1.30516087]      0.273451
26        [0.69483913 0.90474389 1.14267561]      0.312335
27        [0.85732439 0.69483913 0.83477564]      0.359141
28        [1.16522436 0.85732439 0.74900088]      0.264241
29        [1.25099912 1.16522436 0.95995772]      0.246123
30        [1.04004228 1.25099912 1.19381595]      0.186937
31        [0.80618405 1.04004228 1.18068017]      0.198071
32        [0.81931983 0.80618405 0.96208954]      0.243073
33        [1.03791046 0.81931983 0.81056264]      0.210606
34        [1.18943736 1.03791046 0.89218337]      0.183776
35        [1.10781663 1.18943736 1.08841943]      0.164982
36        [0.91158057 1.10781663 1.16223045]      0.168844
37        [0.83776955 0.91158057 1.04240461]      0.188253
38        [0.95759539 0.83776955 0.8869769 ]      0.162310
39        [1.1130231  0.95759539 0.8777115 ]      0.139645
40        [1.1222885  1.1130231  1.00940463]      0.138492
41        [0.99059537 1.1222885  1.11611157]      0.117343
42        [0.88388843 0.99059537 1.07839079]      0.122120
43        [0.92160921 0.88388843 0.95502639]      0.129174
44        [1.04497361 0.92160921 0.89646202]      0.118055
45        [1.10353798 1.04497361 0.96273067]      0.111790
46        [1.03726933 1.10353798 1.06449506]      0.092216
47        [0.93550494 1.03726933 1.08144843]      0.094100
48        [0.91855157 0.93550494 1.00334786]      0.101425
49        [0.99665214 0.91855157 0.92985382]      0.078363
50        [1.07014618 0.99665214 0.9445851 ]      0.072981
```

QUESTION 2:

Testing results of codes for given linear system

b) Gauss-Seidel method.

So, we can see from results below that Gauss-Seidel method **diverges** because solution at each iteration is either [2,2,2]^T or [0,0,0]^T. Hence, this clearly means that solution **does not converge**, or it **diverges**

```
k       x(k)         Rele
1       [2. 2. 2.]   1.000000
2       [0. 0. 0.]   inf
3       [2. 2. 2.]   1.000000
4       [0. 0. 0.]   inf
5       [2. 2. 2.]   1.000000
6       [0. 0. 0.]   inf
7       [2. 2. 2.]   1.000000
8       [0. 0. 0.]   inf
9       [2. 2. 2.]   1.000000
10      [0. 0. 0.]   inf
11      [2. 2. 2.]   1.000000
12      [0. 0. 0.]   inf
13      [2. 2. 2.]   1.000000
14      [0. 0. 0.]   inf
15      [2. 2. 2.]   1.000000
16      [0. 0. 0.]   inf
17      [2. 2. 2.]   1.000000
18      [0. 0. 0.]   inf
19      [2. 2. 2.]   1.000000
20      [0. 0. 0.]   inf
21      [2. 2. 2.]   1.000000
22      [0. 0. 0.]   inf
23      [2. 2. 2.]   1.000000
24      [0. 0. 0.]   inf
25      [2. 2. 2.]   1.000000
26      [0. 0. 0.]   inf
27      [2. 2. 2.]   1.000000
28      [0. 0. 0.]   inf
29      [2. 2. 2.]   1.000000
30      [0. 0. 0.]   inf
31      [2. 2. 2.]   1.000000
32      [0. 0. 0.]   inf
33      [2. 2. 2.]   1.000000
34      [0. 0. 0.]   inf
35      [2. 2. 2.]   1.000000
36      [0. 0. 0.]   inf
37      [2. 2. 2.]   1.000000
38      [0. 0. 0.]   inf
39      [2. 2. 2.]   1.000000
40      [0. 0. 0.]   inf
41      [2. 2. 2.]   1.000000
42      [0. 0. 0.]   inf
43      [2. 2. 2.]   1.000000
44      [0. 0. 0.]   inf
45      [2. 2. 2.]   1.000000
46      [0. 0. 0.]   inf
47      [2. 2. 2.]   1.000000
48      [0. 0. 0.]   inf
49      [2. 2. 2.]   1.000000
50      [0. 0. 0.]   inf
```