Final Project CCOM-4065 Numerical Linear Algebra

Ian Flores Siaca (801-13-2348)

February 24, 2018

```
In [1]: import numpy as np
        from scipy.linalg import qr
        from scipy.linalg import solve
        from numpy.linalg import norm
        from scipy.linalg import hilbert
        from scipy.linalg import lu_factor
        from scipy.linalg import lu_solve
        from tabulate import tabulate
```

1 Problem 1

```
In [2]: ## Generate a 10 by 10 matrix with random integers from 0 to 100.
        A = np.random.randint(100,size=(10,10))

In [3]: A

Out[3]: array([[88,  1, 46, 89, 38, 70, 39, 20, 49, 13],
               [19, 51, 12, 81, 71, 72,  5,  0, 71, 87],
               [17, 36, 17, 78, 53, 69,  6, 83, 19, 81],
               [14, 13,  8, 62,  4, 78, 32, 57, 33, 64],
               [10,  7, 41, 53, 85,  3, 91, 26, 95,  6],
               [94, 30, 18, 73, 83, 81, 79,  5, 51, 30],
               [23,  4, 13, 21, 37, 99, 24, 35, 71, 51],
               [24, 25, 97, 20, 87, 11, 42, 14, 73,  5],
               [76, 66, 93, 91, 14, 59, 96, 82, 50, 42],
               [99, 91,  8, 42, 35, 25, 93, 59, 41, 30]])

In [4]: ## Add 0.1 in the diagonal
        L = np.tril(A, -1) + np.diag(np.repeat(0.1, 10))

In [5]: L

Out[5]: array([[ 0.1,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ],
               [19. ,  0.1,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ],
               [17. , 36. ,  0.1,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ],
               [14. , 13. ,  8. ,  0.1,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ],
               [10. ,  7. , 41. , 53. ,  0.1,  0. ,  0. ,  0. ,  0. ,  0. ],
```

```
            [94. , 30. , 18. , 73. , 83. ,  0.1,  0. ,  0. ,  0. ,  0. ],
            [23. ,  4. , 13. , 21. , 37. , 99. ,  0.1,  0. ,  0. ,  0. ],
            [24. , 25. , 97. , 20. , 87. , 11. , 42. ,  0.1,  0. ,  0. ],
            [76. , 66. , 93. , 91. , 14. , 59. , 96. , 82. ,  0.1,  0. ],
            [99. , 91. ,  8. , 42. , 35. , 25. , 93. , 59. , 41. ,  0.1]])
```

```
In [6]: print("L Determinant: " + str(np.linalg.det(L)))
```

```
L Determinant: -6.314939192258991e-06
```

```
In [7]: L_T = np.transpose(L)
```

```
In [8]: L_T
```

```
Out[8]: array([[ 0.1, 19. , 17. , 14. , 10. , 94. , 23. , 24. , 76. , 99. ],
            [ 0. ,  0.1, 36. , 13. ,  7. , 30. ,  4. , 25. , 66. , 91. ],
            [ 0. ,  0. ,  0.1,  8. , 41. , 18. , 13. , 97. , 93. ,  8. ],
            [ 0. ,  0. ,  0. ,  0.1, 53. , 73. , 21. , 20. , 91. , 42. ],
            [ 0. ,  0. ,  0. ,  0. ,  0.1, 83. , 37. , 87. , 14. , 35. ],
            [ 0. ,  0. ,  0. ,  0. ,  0. ,  0.1, 99. , 11. , 59. , 25. ],
            [ 0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0.1, 42. , 96. , 93. ],
            [ 0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0.1, 82. , 59. ],
            [ 0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0.1, 41. ],
            [ 0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0.1]])
```

```
In [9]: print("L Transpose Determinant: " + str(np.linalg.det(L_T)))
```

```
L Transpose Determinant: 9.999999999999996e-11
```

```
In [10]: L_L_T = np.matmul(L, L_T)
         print("The Determinant of L times L Transpose: " + str(np.linalg.det(L_L_T)))
```

```
The Determinant of L times L Transpose: 147646044694.37805
```

```
In [11]: print("The rank of L is: " + str(np.linalg.matrix_rank(L)))
```

```
The rank of L is: 9
```

1.1 Explanation

The calculated values distance from the theorethical values. The theorethical values of the determinants of L and L^T are calculated multiplying the 10 elements in the diagonal. However, the algorithm that the NumPy library uses is the recursive algorithm to calculate the determinant. This makes the values differ. We could achieve erroneous conclusions about the condition number of these matrices and about their invertibility depending on the algorithm used for the determinant.

2 Problem 2

```
In [12]: values = []
         for i in [2,5,10,12,13,14]:
             matrix = hilbert(i)
             determinant = np.linalg.det(matrix)

             ## Solution ##
             b = np.matmul(np.ones(i), matrix)

             #### LU #####
             LU, P = lu_factor(matrix)
             x_lu = lu_solve((LU, P), b)
             x_lu = max(abs(1-x_lu))

             #### QR #####
             q, r = qr(matrix)
             y = np.matmul(np.transpose(q), b)
             x_qr = solve(r, y)
             x_qr = max(abs(1-x_qr))
             values.append([i, determinant, x_lu, x_qr])
```

```
/usr/local/lib64/python3.6/site-packages/scipy/linalg/basic.py:40: RuntimeWarning: scipy.linalg.
Ill-conditioned matrix detected. Result is not guaranteed to be accurate.
Reciprocal condition number/precision: 3.7780862126689677e-17 / 1.1102230246251565e-16
  RuntimeWarning)
/usr/local/lib64/python3.6/site-packages/scipy/linalg/basic.py:40: RuntimeWarning: scipy.linalg.
Ill-conditioned matrix detected. Result is not guaranteed to be accurate.
Reciprocal condition number/precision: 1.0343900256763508e-18 / 1.1102230246251565e-16
  RuntimeWarning)
/usr/local/lib64/python3.6/site-packages/scipy/linalg/basic.py:40: RuntimeWarning: scipy.linalg.
Ill-conditioned matrix detected. Result is not guaranteed to be accurate.
Reciprocal condition number/precision: 1.9230260898501144e-18 / 1.1102230246251565e-16
  RuntimeWarning)
```

```
In [13]: print(tabulate(values, headers=["n", "Determinant", "max |1-x| (LU)", "max |1-x| (QR)"]
```

n	Determinant	max |1-x| (LU)	max |1-x| (QR)
2	0.0833333	6.66134e-16	1.44329e-15
5	3.7493e-12	2.77822e-12	3.15667e-11
10	2.16421e-53	0.00016919	0.000135915
12	2.55055e-78	0.287098	0.0285862
13	2.45182e-92	0.251896	3.86957
14	-2.38717e-106	3.57664	8.64639

3 Problem 3

The code is included in the letter_A.ps file as an annex in PostScript format.

4 Problem 4

a) Write a function that given an adjacency matrix A and a q values, builds a Google Matrix G from the following equation.

$$G_{ij} = \frac{q}{n} + (1-q)\frac{A_{ji}}{n_j}$$

```
In [14]: A = np.array([[0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
                       [0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
                       [0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0],
                       [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
                       [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
                       [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0],
                       [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0],
                       [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
                       [0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0],
                       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
                       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
                       [0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0],
                       [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0],
                       [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1],
                       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0]
                      ])
```

```
In [15]: def Google(A, q):
             G = np.zeros(A.shape)
             size = A.shape[0]

             for i in range(size):
                 for j in range(size):
                     G[i][j] = (q/size) + (1-q)*(A[j][i]/sum(A[j]))

             return(G)
```

b) Using the Potence Method, for the graph given in Figure 1, with q = 0, q= 0.15, and q = 0.5, calculate the rankings for each page. Remember that you will calculate the eigenvector asociated to the eigenvalue 1 of the G matrix and that you will normalize the result as follows

$$p = \frac{p_i^{(M)}}{\sum p_i^M}$$

```
In [16]: new_x = []
         def iterative(G, Tol):
             MaxItr = 1000
             Err = 1
             x = np.repeat(0.1, G.shape[1])
             xn = x
             it = 0
             while (Err > Tol):
                 it = it + 1
                 if (it > MaxItr):
                     print("El metodo excede 1000 iteraciones")
                 else:
                     u = np.divide(x,sum(x))
                     xn = np.matmul(G, u)
                     lam = np.matmul(np.transpose(u), xn)
                     Err = norm(x-xn, np.inf)
                     x = xn
             #print("The page with highest probability is Page " + str(np.argmax(x+1)) + "\n")



             x_index = np.argsort(x)
             x_index = x_index.tolist()
             x_index = [i+1 for i in x_index]

             x = np.sort(x, axis=0)
             values = []

             for i in range(G.shape[1]):
                 if (G.shape[1] < 15):
                     if (x_index[i] >= 10):
                         temp_x_index = int(x_index[i]) + 1
                         values.append([temp_x_index, x[i]])
                     else:
                         values.append([x_index[i], x[i]])
                 else:
                     values.append([x_index[i], x[i]])

             print(tabulate(values, headers=["Page", "Probability"]))

In [17]: G_0 = Google(A, 0)
         iterative(G_0, 0.000005)
```

```
  Page    Probability
------  -------------
     2      0.0115831
     3      0.0115831
     1      0.0154445
```

```
     4      0.0154445
     5      0.0308871
     6      0.0308871
     7      0.0308871
     8      0.0308871
     9      0.0810807
    12      0.0810807
    10      0.110038
    11      0.110038
    14      0.146717
    13      0.146721
    15      0.146721
```

```
In [18]: G_15 = Google(A, 0.15)
         iterative(G_15, 0.000005)
```

```
  Page    Probability
------  -------------
     4      0.0268248
     1      0.0268248
     3      0.0298608
     2      0.0298608
     5      0.0395877
     6      0.0395877
     7      0.0395877
     8      0.0395877
     9      0.0745633
    12      0.0745633
    10      0.106322
    11      0.106322
    14      0.116326
    13      0.125091
    15      0.125091
```

```
In [19]: G_50 = Google(A, 0.5)
         iterative(G_50, 0.000005)
```

```
  Page    Probability
------  -------------
     1      0.0467355
     4      0.0467355
     5      0.0536094
     6      0.0536094
     7      0.0536094
     8      0.0536094
     2      0.0540208
     3      0.0540208
```

```
     9       0.0676356
    12       0.0676356
    14       0.0785699
    13       0.0904709
    15       0.0904709
    10       0.0946335
    11       0.0946335
```

c) Study what would happen with the page rankings if we eliminate page 10 from the graph (all the links to and from page 10 are eliminated from the graph because it no longer exists). Compare with the results from part (b).

```
In [20]: A_10 = np.delete(A, 9, 0)
         A_10 = np.delete(A_10, 9, 1)
```

```
In [21]: G_0 = Google(A_10, 0)
         iterative(G_0, 0.000005)
```

```
  Page    Probability
------  -------------
     2      0.0178042
     3      0.0181985
     1      0.023473
     5      0.0234738
     6      0.0236052
     4      0.0245292
     9      0.03508
    13      0.0466831
     7      0.0489255
     8      0.0490568
    12      0.12897
    14      0.140048
    11      0.186735
    15      0.233418
```

```
In [22]: G_15 = Google(A_10, 0.15)
         iterative(G_15, 0.000005)
```

```
  Page    Probability
------  -------------
     4      0.0320698
     3      0.0359359
     2      0.0409132
    13      0.0411611
     6      0.0413913
     5      0.0428014
     1      0.0470957
```

```
      9      0.0482246
      8      0.0502483
      7      0.0516583
     12      0.103598
     14      0.107461
     11      0.170961
     15      0.18648
```

```
In [23]: G_50 = Google(A_10, 0.5)
         iterative(G_50, 0.000005)
```

```
  Page    Probability
------  -------------
    13      0.0485539
     4      0.0503148
     8      0.058404
     3      0.0586594
     7      0.0589934
     6      0.0616309
     2      0.0621974
     5      0.0622203
     9      0.0645587
     1      0.0668253
    14      0.0770388
    12      0.0774797
    15      0.116744
    11      0.13638
```

5 Problem 5

Functions for the Iterative Methods in this Problem

```
In [45]: def jacobi(A, b, x0):
             n = A.shape[0]
             xn = np.zeros(n)
             k = 0
             while k < 5:
                 k = k + 1
                 for i in range(n):
                     if abs(A[i,i]) < np.finfo(float).eps:
                         print("Cero en la diagonal")
                     else:
                         under_diag = i-1
                         upper_diag = i+1
                         if (under_diag > 2 and upper_diag < n):
                             xn[i] = (b[i] - A[i, i-1]*x0[i-1] - A[i, i+1]*x0[i+1])/A[i,i]
```

```
            err = norm(xn-x0, np.inf)
            x0 = xn
        print("Mean Predicted X: " + str(sum(x0)/len(x0)))
        print("Iterations: " + str(k))
```

```
In [46]: def seidel(A, b, x0):
             n = A.shape[0]
             xn = np.zeros(n)
             it = 0
             while it < 5:
                 it = it + 1
                 for i in range(n):
                     if abs(A[i,i] < np.finfo(float).eps):
                         print("Cero en la diagonal")
                     else:
                         under_diag = i-1
                         upper_diag = i+1
                         if (under_diag > 2 and upper_diag < n):
                             xn[i] = (b[i] - A[i, i-1]*xn[i-1] - A[i, i+1]*x0[i+1])/A[i,i]
                 err = norm(xn-x0, np.inf)
                 x0 = xn
             print("Mean Predicted X: " + str(sum(x0)/len(x0)))
             print("Iterations: " + str(it))
```

```
In [47]: def sor(A, b, x0, w):
             n = A.shape[0]
             xn = np.zeros(n)
             it = 0
             while it < 5:
                 it = it + 1
                 for i in range(n):
                     if abs(A[i,i] < np.finfo(float).eps):
                         print("Cero en la diagonal")
                     else:
                         under_diag = i-1
                         upper_diag = i+1
                         if (under_diag >= 2 and upper_diag < n):
                             xn[i+1] = x0[i] + w*(((b[i] - A[i, i-1]*xn[i-1] - A[i, i+1]*x0[i+1]
                 err = norm(xn-x0, np.inf)
                 x0 = xn

             print("Mean Predicted X: " + str(sum(x0)/len(x0)))
             print("Iterations: " + str(it))
```

N = 100

```
In [48]: N = 100
         diagonals = np.zeros((3, N))
```

```
diagonals[0,:] = -1
diagonals[1, :] = 3
diagonals[2, :] = -1
import scipy.sparse
A = scipy.sparse.spdiags(diagonals, [-1, 0, 1], N, N, "csr")

b = np.repeat(1, N)
b[0] = 2
b[-1] = 2

x0 = np.repeat(0, N)
```

```
In [49]: jacobi(A, b, x0)
```

```
Mean Predicted X: 0.8997491807651269
Iterations: 5
```

```
In [50]: seidel(A,b,x0)
```

```
Mean Predicted X: 0.9093009116369456
Iterations: 5
```

```
In [51]: w = 1.2
         sor(A, b, x0, w)
```

```
Mean Predicted X: 0.9260069444436325
Iterations: 5
```

N = 100,000

```
In [52]: N = 100000
         diagonals = np.zeros((3, N))
         diagonals[0,:] = -1
         diagonals[1, :] = 3
         diagonals[2, :] = -1
         import scipy.sparse
         A = scipy.sparse.spdiags(diagonals, [-1, 0, 1], N, N, "csr")

         b = np.repeat(1, N)
         b[0] = 2
         b[-1] = 2

         x0 = np.repeat(0, N)
```

```
In [53]: jacobi(A, b, x0)
```

```
Mean Predicted X: 0.9582747491798954
Iterations: 5
```

```
In [54]: seidel(A,b,x0)
```

```
Mean Predicted X: 0.9686905509116369
Iterations: 5
```

```
In [55]: w = 1.2
         sor(A, b, x0, w)
```

```
Mean Predicted X: 0.979113506945399
Iterations: 5
```

Explanation: I had problems with the infinite norm, given that always at the second iteration it was lower than the epsilon of the machine, so I changed the problem to one in which all the methods would iterate the same amount of time and then evaluate the mean of the predicted X's and use that as a metric. With this method, a w of 1.2 could improve the performance of the SOR method over the Gauss-Seidel Method.