Name: Yousif Aldossary

ID: 300231539

Subject: Math245

Assignment type: Project

Due Date: 22/10/2019

Title: Numrical linear algebra: Cholesky and BandCholesky for banded coefficents and linear systems

General

Intro:

In this final project we will mainly focus on Chelosky method of matrix decompesition for linear algebra. We will be using 4 types of Chelosky methods:

1). Cheloskyfacotrization
2). Band Chelosky
3). Forward Chelosky
4). Backward Chelosky

This problem will have a banded coeefficient with upperwidth of U = 0 and so is the lowerwidth L = 0. Therefore we will be working with the diagonal of the matrices given by file matrix1.mtx in the Assignment.
However, A = L.tranpose(L) for forward and backward substitution of the band matrices and the given function is Ax = b for us to solve.

There are few things should be taken under consideration such as:
it is symmetric, positive definite, p = 48 and none-zero diagonal = 97, and lastly a total elements of 3648 in the given matrix

The purpose of this data construction is studying the structural stability of an oil rig we need to solve a large sparse eigenvalue problem involving a banded.

Q1

Intro

In this question we will be working on two ideas BandCholesky and Cholesky and understnad the difference between them. However, they both use the same math equation its only the implementation that differ. Where

$$Ax = b$$

Forward substitution

$$Ux = y$$

Backward substitution therefore:

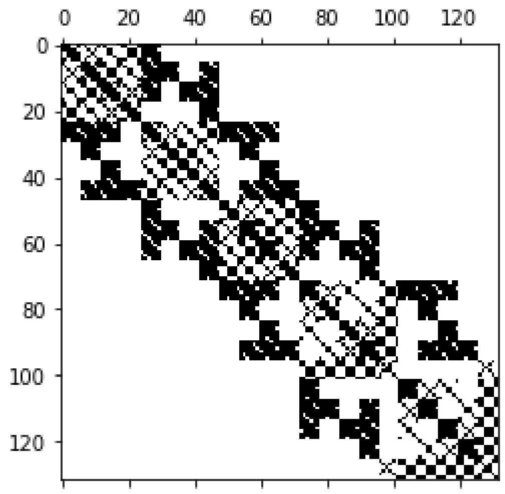
$$b = Uy$$

Moreover to draw the shape of the matrix we need to save the python file with the matrix.mtx (where we extract the data from) together in same folder to make the system implement the data. For the program to implement the file data matrix.mtx, we need to use mmread from scipy.io given in question one.
Lastly, the codes to write to be able to use cholesky is also given in introduction of the last project called band cholesky.

In [331]:

```
import numpy as np
import numpy.linalg as npl
import scipy.io as sc
import scipy.sparse as sp
import matplotlib.pyplot as plt

def BandCholesky(A,p):
    n = len(A)
    for j in range(0,n):
        for k in range (max(1, j+1-p)-1,j):
            lamd = min(k+p+1,n)
            A[j:lamd,j]= A[j:lamd,j] - A[j,k]* A[j:lamd,k]

        lamd =  min(j+p+1,n)
        A[j:lamd,j] =  A[j:lamd,j]/np.sqrt(A[j,j])
    return np.tril(A)

B = sc.mmread("matrix1(1).mtx")
A = B.toarray()
#A = np.array([[2.0, -1.0, 0.0], [-1.0, 3, -1.0], [0.0, -1.0, 2.0]])
L=BandCholesky(A,1)

#A = np.array([[2.0, -1.0, 0.0], [-1.0, 3, -1.0], [0.0, -1.0, 2.0]])
#print(npl.cholesky(A))
#L = sp.lil_matrix(A)
# "tril return lower triangular, triu return upper traingular"
# print(A)

print(L)

plt.spy(A)
plt.show()

#s=BandCholesky(A)
#print(s)
#p = 48
#print(BandCholesky(A,p))
```

```
[[  44.67691594    0.            0.          ...    0.
     0.            0.        ]
 [  12.70265881   42.83303955    0.          ...    0.
     0.            0.        ]
 [ 775.41907594   18.10329325   86.42762712 ...    0.
     0.            0.        ]
 ...
 [    0.            0.            0.          ... 1033.18396194
     0.            0.        ]
 [    0.            0.            0.          ...    0.
  1033.18396194    0.        ]
 [    0.            0.            0.          ...    0.
     0.         1216.75098558]]
```



In [103]:

```
# Forward substitution for Cholesky
import scipy.sparse as sp
import numpy as np
from numpy import linalg
def FCholesky(A,b):
    n=len(A)
    for j in range(0,n):
        b[j]=b[j]/A[j,j]
        b[j+1:np.min(j+p+1,n)]= b[j+1:np.min(j+p+1,n)]-np.dot(A[j+1:np.min(j+p+1,n),j],b[j]
    return b
```

In [84]:

```
# Backward substitution for Cholesky
import numpy as np
def BCholesky(A,b):
    n=len(A)
    for j in range(0,n):
        b[j]=b[j]/A[j,j]
        b[np.max(1,j+1-p):j]= b[np.max(1,j+1-p):j]-np.dot(np.transpose(A[j,np.max(1,j+1-p):
    return b
```

Observation

As shown above the program works perfectly and it is able to read all the data from the document given by Dr Dimitrios though blackboard. I was able to confirm this using spy from matplotlib.pyplot since it resulted the same shape given in the project sheet. Therefore I am able to confirm that BandChelosky is functioning.

Conclusion:

Now the function are written for BandCholesky as required for Q1, now we need to compare it to the regular Cholesky given in lesson 18.

Q2

I am computing LD(L^t) for A factorization

Intro:

In this question we will compare the results of both Cholesky and BandCholesky, but first we need to implement both codes and factorize them.
Cholesky is implementation is given in Lesson 18.
The idea of factorization is to use L (as lower triangular matrix) x D (the diagonal of the same matrix) x Transposed(L). This is how to factorize the Cholesky mathmetically.
However, in this situation we will write the code for general cholesky then replace A for LxDx(transpoe(L)) in the input section

In [600]:

```
import numpy as np
import numpy.linalg as npl
import matplotlib.pyplot as plt

def cholesky(A):

    n = len(A)

    for i in range(0,n):
        try:
            A[i,i] = np.sqrt(A[i,i] - np.dot(A[i,0:i],A[i,0:i]))
        except ValueError:
            error.err("Matrix is not positive definite")
        for j in range(i+1,n):
            A[j,i] = (A[j,i]-np.dot(A[j,0:i],A[i,0:i]))/A[i,i]

    for k in range(1,n):
        A[0:k,k]=0.0

    return A
#B = sc.mmread("matrix1(1).mtx")
#A = B.toarray()
A = np.array([[2.0, -1.0, 0.0], [-1.0, 2.0, -1.0], [0.0, -1.0, 2.0]])
L = cholesky(np.tril(A)*np.diagonal(A)*np.transpose(np.tril(A)))
print(L)
```

```
[[ 2.82842712  0.          0.        ]
 [-0.          2.82842712  0.        ]
 [ 0.          0.          2.82842712]]
```

In [598]:

```
import numpy as np
import numpy.linalg as npl
import matplotlib.pyplot as plt

def BandCholesky(A,p):
    n = len(A)
    for j in range(0,n):
        for k in range (max(1, j+1-p)-1,j):
            lamd = min(k+p+1,n)
            A[j:lamd,j]= A[j:lamd,j] - A[j,k]* A[j:lamd,k]

        lamd =  min(j+p+1,n)
        A[j:lamd,j] =  A[j:lamd,j]/np.sqrt(A[j,j])
    return np.tril(A)
#B = sc.mmread("matrix1(1).mtx")
#A = B.toarray()
A = np.array([[2.0, -1.0, 0.0], [-1.0, 2.0, -1.0], [0.0, -1.0, 2.0]])
L = BandCholesky(np.tril(A)*np.diagonal(A)*np.transpose(np.tril(A)), 1)
print(L)
```

```
[[ 2.82842712  0.          0.        ]
 [-0.          2.82842712  0.        ]
 [ 0.         -0.          2.82842712]]
29.7
```

Observation:

I was unable to tell the difference in a large matrix. Therefore I had to use 3x3 matrix that is symmetric and definite positive, because in cholesky these two condition must be met for guassian elimination. In addition both codes are different from each others. However, they both share the same results in 3x3 matrix and if we look at the first and last few digits in large matrix, they also share same values. Therefore, we can assume that both function results in the same way but different python set ups.

Conslution:

Although, the function accuracy is the same, but is there a function to construct Cholesky, and is it as accurate as the results above.

Q3

In this question, we will test the accuracy of both functions of cholesky and Band Cholesky using the code numpy.linalg.cholesky. Also, we will use the same idea to factorize Cholesky then add it to the python function cholesky and compare the results from Q2

In [591]:

```
import numpy as np
import numpy.linalg as npl

#B = sc.mmread("matrix1(1).mtx")
#A = B.toarray()
A = np.array([[2.0, -1.0, 0.0], [-1.0, 2.0, -1.0], [0.0, -1.0, 2.0]])
g = np.tril(A)*np.diagonal(A)*np.transpose(np.tril(A))
L = npl.cholesky(g)
print(L)
```

```
[[ 2.82842712  0.          0.        ]
 [-0.          2.82842712  0.        ]
 [ 0.          0.          2.82842712]]
```

observation:

The function resulted the same in accuracy compared to Q2. I had to use the same 3x3 matrix since I am unable to tell the difference in large matrices, and sharing the same idea of definite positive and symmetrix matrix.

Conclusion:

The function is working perfectly so far. but we have not calculated x in both Ux=y or b= xA. Therefore, I can conclude that in Q1 to Q3 I have only tested the acuracy of the function through the accuracy of the results.

Q4

In the pervious questions we test the A = LDL^t, but in this question we will move forward to calulcate b and x for the formula b = xA, and then use the norm of the function using ||x(numerical solution) -x(exact solution) || using numpy function norm.
I will have to adjust/imporve the code slightly to find b and x. However, xexacts are all 1 therefore we use the function numpy.ones() with the same size of A matrix. I will find the results of b and x in both functions of Cholesky and BandCholesky but I will rename the function to solve and Bandsolve as required in the Project assignment

In [602]:

```
# Q4 Solution
def solve(L,b):
    n = len(A)

    # Solution of Ly=b

    for k in range(0,n):
        b[k] = (b[k] - np.dot(L[k,0:k],b[0:k]))/L[k,k]

    # Solution of L^T x =y

    for k in range(n-1,-1,-1):
        b[k] = (b[k] - np.dot(L[k+1:n,k],b[k+1:n]))/L[k,k]

    return b

#B = sc.mmread("matrix1(1).mtx")
#A = B.toarray()
A = np.array([[2.0, -1.0, 0.0], [-1.0, 2.0, -1.0], [0.0, -1.0, 2.0]])
n = len(A)

xexact = np.ones(n)
b = np.dot(A,xexact)
L = cholesky(A)

x1 = solve(L,b) #Forward substitution finding b = Lx
print(x1)
x2 = solve(np.transpose(L),x1) #Backward substitution find y = Ux
print(x2)

xnorm = npl.norm(x1-xexact)
print(xnorm)
xnorm2 = npl.norm(x2-xexact)
print(xnorm2)
```

```
[1. 1. 1.]
[0.5        0.66666667 0.75      ]
0.6508541396588878
0.6508541396588878
```

In [588]:

```
def SCholesky(A,b,p):
    n=len(A)
    for j in range(0,n):
        b[j]=b[j]/A[j,j]
        b[j+1:min(j+p+1,n)]= b[j+1:min(j+p+1,n)]-A[j+1:min(j+p+1,n),j]*b[j]

    for j in range(n-1,-1,-1):
        b[j]=b[j]/A[j,j]
        b[max(0,j-p):j]= b[max(0,j-p):j]-np.dot(np.transpose(A[j,max(0,j-p):j]),b[j])
    return b

B = sc.mmread("matrix1(1).mtx")
A = B.toarray()
#A = np.array([[2.0, -1.0, 0.0], [-1.0, 2.0, -1.0], [0.0, -1.0, 2.0]])
n = len(A)

xexact = np.ones(n)
b = np.dot(A,xexact)
p = 48

L = BandCholesky(A,p)

x1 = SCholesky(L,b,p) #Forward substitution finding b = Lx
print(x1)
x2 = SCholesky(np.transpose(L),x1,p) #Backward substitution find y = Ux
print(x2)

xnorm = npl.norm(x1-xexact)
print(xnorm)
xnorm2 = npl.norm(x2-xexact)
print(xnorm2)
```

```
[ 2.29536436e-03 -1.77668712e-03  1.05674124e-04  2.39888204e-07
  2.09034568e-07 -4.09113187e-06  8.54604965e-04 -4.55744061e-03
 -2.81079859e-04  4.11359673e-06  1.14232361e-06 -4.55309483e-07
  4.29648635e-03 -1.05110476e-03  3.97190268e-04  1.19813330e-06
  3.61593769e-06 -1.43472193e-07  4.56149203e-03 -3.99781510e-03
  3.37396845e-05  1.53135322e-06  1.81965362e-06  1.58407409e-06
  9.11743286e-04 -8.30912363e-04  4.61903212e-05  2.43712562e-06
  2.80297768e-06 -4.70261286e-06  1.05348720e-03 -5.03334842e-03
  7.67538213e-05  4.45111602e-06  7.70957635e-07 -4.10234749e-07
  4.43710966e-03 -1.19535227e-03 -3.11062914e-06  7.84874203e-07
  4.24259009e-06  3.02776571e-08  2.85224395e-03 -2.49587294e-03
  3.00268550e-05  5.91346424e-06  6.21338025e-06  2.81128257e-06
  3.39242871e-03 -2.85626817e-03  1.47810713e-05  8.73168508e-07
  6.44494939e-07 -3.68568211e-06  6.82925083e-04 -4.76692898e-03
  1.67058014e-04  3.01082797e-06  8.07566009e-07 -4.66878106e-07
  4.55348942e-03 -8.56046999e-04 -8.33892620e-05  1.35667118e-06
  3.08352014e-06 -4.11392342e-07  8.01491293e-03 -7.41274717e-03
 -6.88194299e-06  2.42557155e-06  2.55357927e-06  2.00763653e-06
  1.51950617e-04 -3.97082070e-04  6.78470860e-07 -1.96742727e-06
 -2.23431646e-06  7.63544435e-08  4.89268903e-05 -3.17458795e-03
  6.45775023e-04  3.62795585e-07 -1.34780439e-06  5.27596292e-07
  3.23673017e-03 -7.34569987e-04 -6.16447141e-04 -1.46986081e-06
  2.85160500e-07  4.75829553e-07  2.01157384e-03 -1.88756355e-03
  2.15315225e-05 -3.09926221e-07 -3.85029262e-07 -4.22376909e-07
```

```
   6.47338578e-04 -8.24598030e-04  3.50091731e-06  8.56880049e-07
   8.95740140e-07  8.79411490e-07 -2.76088694e-03  2.33032168e-03
  -1.36622083e-05  2.96797844e-07  2.89163039e-07  1.67383518e-06
  -1.41327891e-03 -9.99216864e-05  5.98311134e-04  1.78669338e-07
   1.79232773e-07  4.11110924e-07  7.28134823e-05  1.51691160e-03
  -5.80465630e-04  1.75286628e-07  1.69036770e-07  5.03287559e-07
  -2.91614563e-04  1.47481465e-04  2.93027405e-05  3.18500611e-07
   3.17464080e-07  2.82127400e-07 -3.07932812e-04  3.95530526e-04
   3.01495451e-06  9.27558068e-07  9.13491974e-07  6.47087512e-07]
[ 5.76127876e-10 -4.45941829e-10  1.73804856e-12  7.87822931e-21
   6.86495723e-21 -9.34924271e-19  9.07558892e-11 -4.83983352e-10
  -3.67370948e-12  9.79230611e-20  2.71927055e-20 -4.74499514e-20
   4.56270973e-10 -1.11623441e-10  5.19127076e-12  2.85212402e-20
   8.60764219e-20 -1.49519147e-20  1.14491745e-09 -1.00343665e-09
   5.54924967e-13  5.02915591e-20  5.97597058e-20  3.61999897e-19
   4.69071777e-11 -4.27486053e-11  4.61158405e-13  4.14803632e-20
   4.77072380e-20 -2.57326028e-19  2.05378673e-10 -9.81257691e-10
   1.26192599e-12  1.41629397e-19  2.45309861e-20 -8.18722382e-20
   8.65020185e-10 -2.33035449e-10 -5.11425188e-14  2.49737953e-20
   1.34994342e-19  6.04263672e-21  1.46741650e-10 -1.28407149e-10
   2.99784375e-13  1.00648339e-19  1.05752969e-19  1.53832816e-19
   1.29630305e-09 -1.09142725e-09  2.43144928e-13  3.25662407e-20
   2.40374878e-20 -1.58464324e-18  4.93062132e-11 -3.44165447e-10
   1.44684851e-12  5.30445635e-20  1.42276433e-20 -3.62478580e-20
   3.28755416e-10 -6.18053676e-11 -7.22213956e-13  2.39017410e-20
   5.43252491e-20 -3.19400097e-20  3.06263062e-09 -2.83253314e-09
  -1.13206242e-13  9.04656387e-20  9.52398952e-20  8.63174728e-19
   7.99296804e-12 -2.08874723e-11  4.90072266e-15 -3.04199832e-20
  -3.45465727e-20  5.30486735e-21  8.56455530e-12 -5.55705338e-10
   1.06068287e-11  1.28207445e-20 -4.76297299e-20  1.70145329e-19
   5.66583209e-10 -1.28585022e-10 -1.01251194e-11 -5.19430517e-20
   1.00772172e-20  1.53450995e-19  1.05813624e-10 -9.92903843e-11
   1.55526238e-13 -4.79201980e-21 -5.95324862e-21 -2.93454234e-20
   4.55641573e-11 -5.80409011e-11  1.92712752e-12  1.07045211e-18
   1.11899784e-18  6.18376342e-19 -3.70173282e-10  3.12444097e-10
  -8.92511042e-13  3.54974861e-20  3.45843514e-20  1.66861276e-18
  -1.05022977e-10 -7.42533757e-12  1.56303902e-11  9.42498655e-21
   9.45470833e-21  1.02640959e-19  5.41088431e-12  1.12724085e-10
  -1.51641909e-11  9.24654521e-21  8.91685898e-21  1.25654452e-19
  -3.90990006e-11  1.97739709e-11  1.91426005e-12  3.80931709e-20
   3.79692001e-20  2.81247154e-19 -1.38344724e-11  1.77699678e-11
   1.61546478e-12  8.14011393e-19  8.01667195e-19  2.95227080e-19]
11.489125293047714
11.489125293047714
```

Observation:

Similarly to the previous questions results, I am unable to tell the difference of a very large matrix. Therefore, I had to set 3x3 matrix following Cholesky conditions. In this results we significantly are looking for x numericals for backward and forward substitution, while x exact are ones with size of the main matrix. Then we find the norms of both backward and forward substitution.
Moreover, the norms for 3x3 and large matrix tests are the same in both bandsolve and solve meaning that the function is working perfectly. However, since I was unable to see the results clearly for x(forward and backward) for large matrix, I had to test it with 3x3 and the results are the same in both functions (solve and bandsolve). As expected x1 resulted all ones, while x2 resulted a slight increase in values. Therefore, we can assume the result will occur the same for the large given matrix.

Consulsion:

The matrix works the way we expected the results of both functions are the same for both 3x3 matrix and large matrix in norms. In conclusion the matrix works following the Cholesky conditions for definite positive and symmetric.

Q5

In this question we will apply the same functions of bandcholesky and cholesky on a new matrix. The new matrix is a conversion of the diagonal of A matrix to rows. The idea to convert is using symmetric as an advantage for band storae. Therefore we are able to convert the matrix to rows. the formula as shown below:

$$Ab(1+i-j,j) = A(i,j),$$
$$for j \leq i \leq min(n, j+p)$$

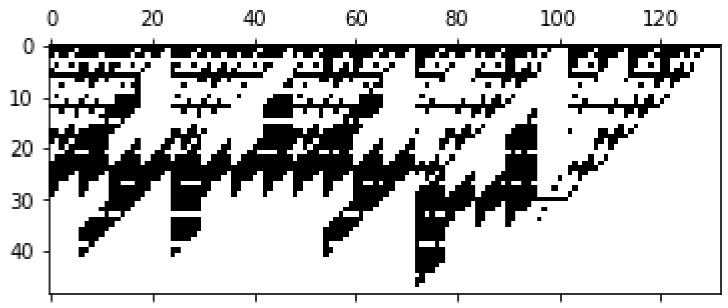
In [565]:

```
import numpy as np
import numpy.linalg as npl
import matplotlib.pyplot as plt

def diagtorow(A,p):
    n = len(A)
    Ab = np.zeros((p+1,len(A)))
    for i in range(n):
        for j in range(n):
            if j<=i<=min(n,j+p):
                Ab[i-j,j]=A[i,j]
            else:
                pass
    return Ab

def BCholesky(A,p):
    n = len(A)
    for j in range(0,n):
        for i in range(0,n):
            for k in range (max(1, j-p)-1,j):
                lamd = min(k+p+1,n)
                A[i-j+1+1-1-1:lamd,j]= A[i-j+1+1-1-1:lamd,j] - A[i-j+1+1-1-1,k]* A[i-j+1+1-

            lamd = min(j+p+1,n)
            A[i-j+1+1-1-1:lamd,j] =  A[i-j+1+1-1-1:lamd,j]/np.sqrt(A[i-j+1+1-1-1,j])
    return np.tril(A)

def DCholesky(A,b,p):
    n=len(A)
    for j in range(0,n):
        for i in range(0,n):
            b[1-j+1-1-1]=b[1-j+1-1-1]/A[i-j+1+1-1-1,j]
            b[1-j+1-1-1:min(j+p,n)]= b[1-j+1-1-1:min(j+p,n)]-A[i-j+1+1-1-1:min(j+p,n),j]*b[

    for j in range(n-1,-1,-1):
        for i in range(0,n):
            b[i-j+1+1-1-1,j]=b[i-j+1+1-1-1,j]/A[i-j+1+1-1-1,j]
            b[j,max(0,j-p):i-j+1+1-1-1]= b[j,max(0,j-p):i-j+1+1-1-1]-np.dot(np.transpose(A[
    return b

B = sc.mmread("matrix1(1).mtx")
A = B.toarray()
p = 48
#A = np.array([[2.0, -1.0, 0.0], [-1.0, 2.0, -1.0], [0.0, -1.0, 2.0]])
Anew = diagtorow(A,p)
plt.spy(Anew)

#n = len(A)
#xexact = np.ones(n)
#print(xexact)
#b = np.dot(Anew,xexact)
#print(b)
#L = BandCholesky(Anew,p)

#x1 = DCholesky(L,b,p) #Forward substitution finding b = Lx
#print(x1)
#x2 = DCholesky(np.transpose(L),x1,p) #Backward substitution find y = Ux
#print(x2)
#
```

```
#xnorm = npl.norm(x1-xexact)
#print(xnorm)
#xnorm2 = npl.norm(x2-xexact)
#print(xnorm2)
```

Out[565]:

```
<matplotlib.image.AxesImage at 0x2ad78a7d630>
```



In [562]:

```
import numpy as np
import numpy.linalg as npl
import matplotlib.pyplot as plt

def diagtorow(A,p):
    n = len(A)
    Ab = np.zeros((p+1,len(A)))
    for i in range(n):
        for j in range(n):
            if j<=i<=min(n,j+p):
                Ab[i-j,j]=A[i,j]
            else:
                pass
    return Ab

def cholesky(A):

    n = len(A)

    for i in range(0,n):
        for j in range(0,n):
            try:
                A[i-j+1+1-1-1,j] = np.sqrt(A[i-j+1+1-1-1,j] - np.dot(A[i-j+1+1-1-1,0:j],A[i
                A[i-j+1+1-1-1,j] = (A[i-j+1+1-1-1,j]-np.dot(A[i-j+1+1-1-1,0:j],A[i-j+1+1-1-
            except ValueError:
                error.err("Matrix is not positive definite")

    for k in range(1,n):
        A[0:k,k]=0.0

    return A

def solve(L,b):
    n = len(A)

    # Solution of Ly=b

    for k in range(0,n):
        b[k] = (b[k] - np.dot(L[k,0:k],b[0:k]))/L[k,k]

    # Solution of L^T x =y

    for k in range(n-1,-1,-1):
        b[k] = (b[k] - np.dot(L[k+1:n,k],b[k+1:n]))/L[k,k]

    return b


A = np.array([[2.0, -1.0, 0.0], [-1.0, 2.0, -1.0], [0.0, -1.0, 2.0]])
Anew = diagtorow(A,1)
#plt.spy(Anew)
#n = len(A)

#xexact = np.ones(Anew)

#b = np.dot(Anew,xexact)

#L = cholesky(Anew)
```

```
#x1 = solve(L,b) #Forward substitution finding b = Lx
#print(x1)
#x2 = solve(np.transpose(L),x1) #Backward substitution find y = Ux
#print(x2)

#xnorm = npl.norm(x1-xexact)
#print(xnorm)
#xnorm2 = npl.norm(x2-xexact)
#print(xnorm2)
```

Observation:

The function is Diagtorow is able to convert diag to row. However, b output is 1x2 matrix, so I am unable to find x norm since BandSolve is unable to multiply or divide unequal column to row, such as 1x2 * 1x3. Therefore I tried to imporve the function but I could not get to the soultion I wanted. However, The issue might not be BandSolve but in fact in BandCholesky function, becase it resulted 1x2 matrix. I have expected the result to be 1x3 matrix to be able to find x for backward and forward substitution. Therefore, I decided to test it on Cholesky function and it happens that I keep getting same errors as before. Therefore looking at the new matrix A it does not meets the requirement to be symmetric nor definite positive. I would conclude that I am unable to find the results of this current issue.

Conclusion:

Despite the many tries I have made to make the function work with bandcholesky and bandsolve. I am unable to obtain results, but I was able to convert the matrix from the diagonal to rows. However, because they are symmetric and definite postive I am able to use the lower triangular to be exact as upper triangular.

Q6

I found both cholesky and bandcholesky have resulted the same, so based on that I would assume the efficiency is the same. Also, the same goes for Solve and BandSolve. The implementation of both bandcholesky and cholesky are quite easy to obtain. However, I would choose BandCholesky as easier implementation since I have not worked on Cholesky but copied it from lesson18. I have tested the CPU at the common functions below and the percentage varies around the percentage shown BandCholesky = 30.0% BandSolve = 24.5% Cholesky = 30.5% Solve = 28.1% Therefore I would assume the average CPU run time it will be 30%.