Project report (5)

Name (300493343)

Introduction

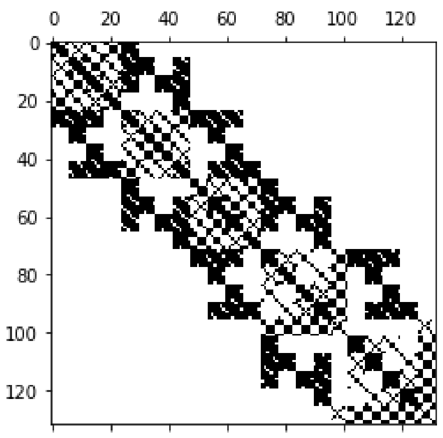
The purpose of this project is to observe and dicuss sparse eigenvalue matrix. The properties of this matrix include being symmetric and positive definite. To observe this sparse matrix, cholesky and band cholesky.

In [165]:

```python
import scipy.io as sio
import scipy.sparse as sparse
from scipy.sparse import linalg
from matplotlib.pyplot import spy
import numpy as np

#1
S = sio.mmread("matrix1.mtx") #import
A = S.toarray() #turning it into a matrix
#A = np.array([[4.0, 5.0, 15.0], [-15.0, 44.25, 22.75], [1.0, 2.75, 3.5]]) example when it
spy(A)
```

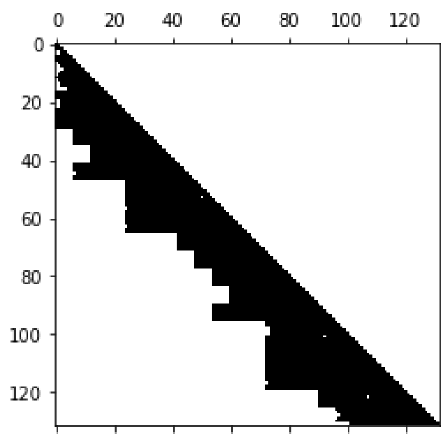
Out[165]:

```
<matplotlib.image.AxesImage at 0x297ea919080>
```



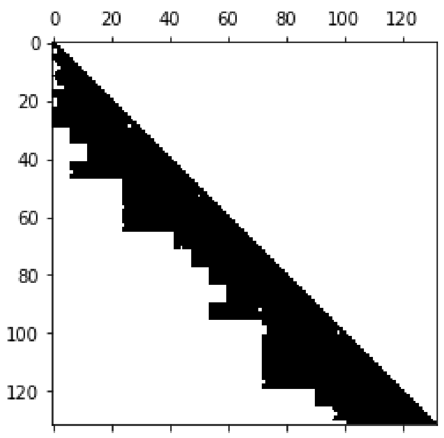
In [166]:
```
#2

def bandCholesky(A,p):
    #p, number of non 0 diaganals above the main diagonals
    #A, matrix
    n = len(A)
    for j in range(n):
        m = max(0, j  - p)
        for k in range(m, j):
            lam = min(k+1 + p,n)
            A[j:lam,j] = A[j:lam,j] - A[j,k]*A[j:lam,k]
        lam = min(j+1+p, n)
        A[j:lam,j] = A[j:lam,j]/np.sqrt(A[j,j])

    return A

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

p = 48
L = np.tril(bandCholesky(A.copy(), p))
spy(L)
```

Out[166]:
```
<matplotlib.image.AxesImage at 0x297ea0889e8>
```



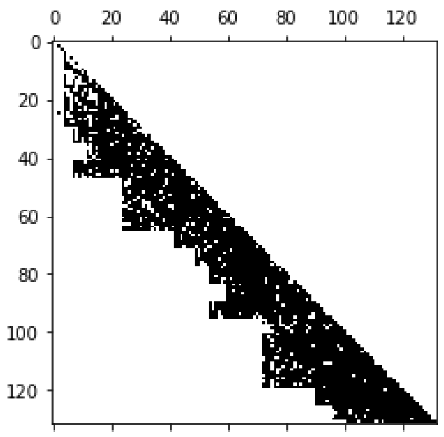
In [167]:
```
C = np.tril(cholesky(A.copy()))
spy(C)
```

Out[167]:
```
<matplotlib.image.AxesImage at 0x297ea953d30>
```



In [168]:
```
spy(abs(C-L))
```

Out[168]:
```
<matplotlib.image.AxesImage at 0x297ea974e48>
```



difference between banded cholesky and normal cholesky, this shows that there is a difference otherwise it would be a blank.

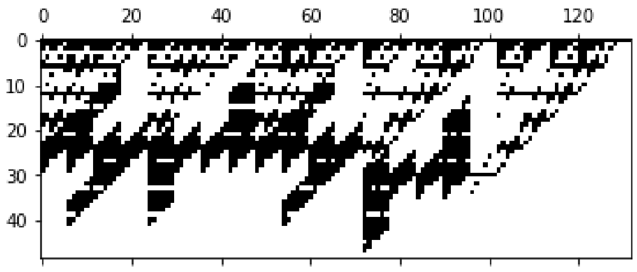
In [170]:
```
#4
import time

from scipy.linalg import solve_banded

def getB(A):
    #j = len(A)
    b = A.sum(axis=1) #summing all rows up
    return b

#i thought band solve is solving


def solve(L,b):

    n = len(b)

    # Solution of Ly=b

    for k in range(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

def solveband(A,b,p):

    n = len(A)

    # Solution of Ly=b

    for k in range(n):#n goes up to n-1 thus we need to do k+1
        b[k] = b[k]/A[k,k]
        m = min(k+1+p,n)
        b[k +1 : m] = b[k + 1: m] - np.dot(A[k+1: m, k], b[k])

    # Solution of L^T x =y

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

    return b

Anew = A.copy()
b = getB(A)
b2 = b.copy()
x = np.repeat(1.0, len(b))

answers = Ab = np.ones(len(A)) #numerical solutions

start = time.time() #timing the solve cholesky method
solve_cholesky = solve(np.tril(cholesky(A.copy())),b)
end = time.time()

solvetime = abs(start - end)

start2 = time.time() #timing the solve band cholesky method
solve_cholesky_band = solveband(np.tril(bandCholesky(A.copy(), p)), b2, 48)
end2 = time.time()
solvebandtime = abs(start2-end2)

print("--solve--")
print(solve_cholesky)
print()
print("--solveband--")
print(solve_cholesky_band)
print()
print("errors of solve(): ",np.linalg.norm(answers - solve_cholesky), "time: ", solvetime)
print("errors of solveband(): ",np.linalg.norm(answers - solve_cholesky_band), "time: ", s
```

```
--solve--
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]

--solveband--
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]

errors of solve():  1.2998045340139598e-12 time:  0.07137560844421387
errors of solveband():  7.172077431340032e-13 time:  0.07489824295043945
```

solve() is for normal cholesky and solveband() is for banded cholesky. Comparing the error of solveband and solve, we can conclude that solveband has less error by 1 order of magnitude.

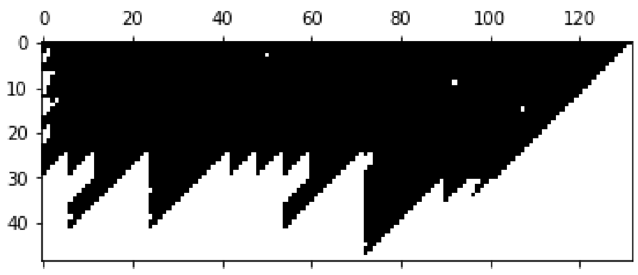
In [171]:
```
#Question 5  part 1
def band(A, p):
    n = len(A)
    print(n)
    Ab = np.zeros((p+1, n)) #determining the dimensions of the banded form
    A = np.tril(A)
    for j in range(n):   # Convert the matrix into banded format
        M = max(1, j-p+1)
        m = min(n, j+p+1)
        for i in range(M-1,m):
            Ab[i-j,j] = A[i,j]
    return Ab

p = 48

Ab = band(A.copy(),p)
spy(Ab)
b=getB(A.copy())
```

```
132
```



In [172]:
```
def bandedCholesky(A,p):
    #p, number of non 0 diaganals above the main diagonals
    #A, matrix
    (a,n) = A.shape
    for j in range(n):
        m = max(0, j  - p)
        for k in range(m, j):
            lam = min(k + p,n)
            A[0:(lam-j),j] = A[0:(lam-j),j] - A[(j-k),k] * A[(j-k):(lam-k),k]
        lam = min(j+p, n)
        A[0:lam-j,j] = A[0:lam-j,j]/np.sqrt(A[0,j])

    return A

aye = bandedCholesky(Ab.copy(), p)
spy(aye)
```

Out[172]:
```
<matplotlib.image.AxesImage at 0x297eba248d0>
```



```
In [164]: def solvebanded(A,b,p):

              n = len(A)

              # Solution of Ly=b

              for k in range(n):#n goes up to n-1 thus we need to do k+1
                  b[k] = b[k]/Ab[0,k]
                  m = min(k+p+1,n)
                  b[k+1:m] = b[k+1:m] - Ab[1:m, k]*b[k]

              # Solution of L^T x =y

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

              return b

          spy(solvebanded(Ab.copy(), b, p))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-164-59d2a27b9e70> in <module>
     20     return b
     21 
---> 22 spy(solvebanded(Ab.copy(), b, p))

<ipython-input-164-59d2a27b9e70> in solvebanded(A, b, p)
      8         b[k] = b[k]/Ab[0,k]
      9         m = min(k+p+1,n)
---> 10         b[k+1:m] = b[k+1:m] - Ab[1:m, k]*b[k]
     11 
     12     # Solution of L^T x =y

ValueError: operands could not be broadcast together with shapes (2,) (48,)
```

Observations and Analysis

I have noticed that banded form solving has a smaller error by a magnitude of 1. However it takes almost twice as long to solve in bandcholesky compared to the normal cholesky. This makes more sense as it has to unpack from banded form before solving. In addition, I also noticed that the difference between band cholesky and cholesky is that cholesky's upper triangle is 0s whereas band cholesky you may get some values along with that, band cholesky takes bands into account where as cholesky doesn't. Nonetheless band cholesky is deem to be more efficient as its error is roughly 10x less than the normal cholesky's method.

A good reason to put a large symmetrical matrix into banded form is to save space. Although it may have a greater time complexity it does save alot of space when working with enormous sets of data. Turning the said banded matrix into banded cholesky will also add to it's accuracy instead of normall solving it as can be seen from comparing the 'error of solve()' and 'error of solvebanded' as well as reaping the benefits of the banded form's low memory usage. However, banded form along with banded cholesky is not reccomended for smaller matrices as it would only a greater time complexity.

Conclusion

In conclusion, banded cholesky gives a greater time complexity in exchange for greater accuracy. Furthermore, putting a symeteric and large matrix in banded storage form will also improve it's storage space as well as having a high degree of accuracy in bandcholesky compared to the normal cholesky's method. However, it's worth metioning that the conditions for cholesky's method is that the matrix is positive definite and symetric. This can prove to be a bit troublesome as some matrix we deal with in the real world may not meet these conditions.