EE 510 - Homework 12 (Using Python)

Name - Onkar Vivek Apte

USC ID - [redacted]

In [53]:
```
import numpy as np
import sympy as sp
from sympy import Matrix
```

Question 1

In [54]:
```
A = np.array([[1,2,4,8],
              [1,3,9,27],
             [1,4,16,64],
             [1,5,25,125]])
b= np.array([[1],
             [-1],
            [1],
            [-1]])
x01= np.array([[0],
             [0],
            [0],
            [0]])
x02= np.array([[1],
             [1],
            [1],
            [1]])
```

Part (i)

In [55]:
```
ainv = np.linalg.inv(A)
ainv
```

Out[55]:
```
array([[ 10.        , -20.        ,  15.        ,  -4.        ],
       [ -7.83333333,  19.        , -15.5       ,   4.33333333],
       [  2.        ,  -5.5       ,   5.        ,  -1.5       ],
       [ -0.16666667,   0.5       ,  -0.5       ,   0.16666667]])
```

Part (ii)

Part (ii)

In [56]:
```
adet = np.linalg.det(A)
adet
```

Out[56]:
```
12.0
```

Part (iii)

In [57]:
```
A_sym = Matrix(A)
print(A_sym.charpoly())
```

```
PurePoly(lambda**4 - 145*lambda**3 + 782*lambda**2 - 410*lambda +
12, lambda, domain='ZZ')
```

Part (iv)

In [58]:
```
evalu,evect = np.linalg.eig(A)
print("EigenValues: \n",evalu)
print("EigenVectors: \n",evect)
```

```
EigenValues:
 [1.39411810e+02 5.00404037e+00 5.53046928e-01 3.11027573e-02]
EigenVectors:
 [[ 0.06604881  0.55133981 -0.94518439  0.72677192]
 [ 0.20166395  0.66311608 -0.17411076 -0.66036048]
 [ 0.4554452   0.48822234  0.27328922  0.18824402]
 [ 0.86460331 -0.13394183 -0.04031029 -0.0170528 ]]
```

Part (v)

In [59]:
```
m = Matrix(A)
J = m.jordan_form(calc_transform = False)
np.array(J).astype(np.cdouble)
```

Out[59]:
```
array([[3.11027573e-02-6.39452411e-34j, 0.00000000e+00+0.00000000e
+00j,
        0.00000000e+00+0.00000000e+00j, 0.00000000e+00+0.00000000e
+00j],
       [0.00000000e+00+0.00000000e+00j, 5.53046928e-01-1.49852222e
-30j,
        0.00000000e+00+0.00000000e+00j, 0.00000000e+00+0.00000000e
+00j],
       [0.00000000e+00+0.00000000e+00j, 0.00000000e+00+0.00000000e
+00j,
        5.00404037e+00+2.15531305e-30j, 0.00000000e+00+0.00000000e
+00j],
       [0.00000000e+00+0.00000000e+00j, 0.00000000e+00+0.00000000e
+00j,
        0.00000000e+00+0.00000000e+00j, 1.39411810e+02+6.52807706e
-31j]])
```

Part (vi)

In [60]:
```
U,sig,V = np.linalg.svd(A)
print("U :",U)
print("\n sig :",sig)
print("\n V :",V)
```

```
U : [[-0.05970473 -0.50955628  0.79149849 -0.33214136]
 [-0.19366886 -0.64293214 -0.12178385  0.73095782]
 [-0.45031148 -0.4151988  -0.54251525 -0.57489715]
 [-0.8695673   0.39319285  0.25372433  0.15772147]]

 sig : [1.46691873e+02 5.58351667e+00 5.63876614e-01 2.59826409e-0
2]

 V : [[-0.01072488 -0.0466931  -0.21082289 -0.97634955]
 [-0.21035029 -0.47331102 -0.83065463  0.20430936]
 [ 0.67554445  0.56077174 -0.47385628  0.06808058]
 [-0.70659566  0.67773999 -0.20255076  0.0190861 ]]
```

Part (vii)

In [61]:
```
anorm = np.linalg.norm(A)
print(anorm)
ainvnorm = np.linalg.norm(ainv)
print(ainvnorm)
#exp(At)
ev,s = np.linalg.eig(A)
e = np.diag(ev)
t = sp.var("t")
e = e*t
print("S")
print(s)
print("e ^\u039Bt")
print(e)
print("S\u207B\u00b9")
print(np.linalg.inv(s),sep="\n")
```

```
146.79918255903198
38.52848873813304
S
[[ 0.06604881  0.55133981 -0.94518439  0.72677192]
 [ 0.20166395  0.66311608 -0.17411076 -0.66036048]
 [ 0.4554452   0.48822234  0.27328922  0.18824402]
 [ 0.86460331 -0.13394183 -0.04031029 -0.0170528 ]]
e ^Λt
[[139.411809943341*t 0 0 0]
 [0 5.00404037108925*t 0 0]
 [0 0 0.553046928294402*t 0]
 [0 0 0 0.0311027572758226*t]]
S⁻¹
[[ 0.00932407  0.04425785  0.21287537  1.03342871]
 [ 0.20599302  0.51262016  0.94559559 -0.63341138]
 [-0.64514827 -0.37920481  1.11958613 -0.45203032]
 [ 0.37979994 -0.88606828  0.71936003 -0.20127887]]
```

Part (viii)

In [62]:
```
gaus = np.linalg.solve(A, b)
print(gaus)
```

```
[[ 49.        ]
 [-46.66666667]
 [ 14.        ]
 [ -1.33333333]]
```

Part (ix)

In [63]:
```
def jacobi(a,b,x,num_iterations):
    #A = L + D + U
    D = np.diag(np.diagonal(a))
    LU = D - a #(-L-U)
    Dinv = np.linalg.inv(D)
    eig = np.linalg.eig(Dinv@LU)[0] #Eig values of Dinv*(-L-U)
    for i in eig:
        if (abs(i)>1):
            print("Jacobi method will not converge") # Eig vals > 1
    for i in range(num_iterations):
        x = np.dot(np.matmul(Dinv, LU), x) + np.dot(Dinv, b)

    return x

print("Jacobi method using x_1 as the initial")
print(jacobi(A,b,x01,50))
print("\nJacobi method using x_2 as the initial")
print(jacobi(A,b,x02,50))
```

```
Jacobi method using x_1 as the initial
Jacobi method will not converge
[[-2.16067413e+13]
 [-1.18978405e+13]
 [-3.51053269e+12]
 [-6.75050659e+11]]

Jacobi method using x_2 as the initial
Jacobi method will not converge
[[1.10020535e+16]
 [6.05832577e+15]
 [1.78754714e+15]
 [3.43732698e+14]]
```

Part (x)

In [64]:
```
def g_seidel(a,b,x,num_iterations):
    DL = np.tril(a) #(D+L)
    U = DL - a #U
    DLinv = np.linalg.inv(DL) #(D+L)inv
    eig = np.linalg.eig(DLinv@U)[0]  #Eig vals of (D+L)inv*(-U)
    for i in eig:
        if (abs(i)>1):
            print("Gauss Seidel method will not converge") #Eig val
    for i in range(num_iterations):
        x = np.matmul(np.matmul(DLinv, U), x) + np.matmul(DLinv,b)

    return x

print("Gauss Seidel using x_1 as the initial")
print(g_seidel(A,b,x01,100))
print("Gauss Seidel using x_2 as the initial")
print(g_seidel(A,b,x02,100))
```

```
Gauss Seidel using x_1 as the initial
[[ 36.50420289]
 [-34.70915436]
 [ 10.4322867 ]
 [ -0.99812479]]
Gauss Seidel using x_2 as the initial
[[ 34.64875226]
 [-32.93363147]
 [  9.9025313 ]
 [ -0.94835102]]
```

Part (xi)

In [65]:
```
=[-1, 0, 1, 2, 3]

ef SOR(A, b, x, w):
 wL = w*np.tril(A,-1)  #Lower triangular
 wU = w*np.triu(A,1)   #Upper triangular
 D = np.diag(np.diagonal(A))    #Diagonal matrix

 for _ in range(100):
     x = np.matmul(np.matmul(np.linalg.inv(D+wL), ((1-w)*D-wU)), x)
 return x

or k in w:
   print("Using x01 and w :",k)
   print(SOR(A, b, x01, k))
   print("Using x02 and w :",k)
   print(SOR(A, b, x02, k))
   print("\n")
```

```
    print('\n')
```

```
Using x01 and w : -1
[[2.28320345e+89]
 [2.38192570e+89]
 [1.30482942e+89]
 [4.97018701e+88]]
Using x02 and w : -1
[[2.21327607e+91]
 [2.30897477e+91]
 [1.26486658e+91]
 [4.81796573e+90]]


Using x01 and w : 0
[[100.        ]
 [-33.33333333]
 [  6.25      ]
 [ -0.8       ]]
Using x02 and w : 0
[[101.        ]
 [-32.33333333]
 [  7.25      ]
 [  0.2       ]]


Using x01 and w : 1
[[ 36.50420289]
 [-34.70915436]
 [ 10.4322867 ]
 [ -0.99812479]]
Using x02 and w : 1
[[ 34.64875226]
 [-32.93363147]
 [  9.9025313 ]
 [ -0.94835102]]


Using x01 and w : 2
[[-123.57719659]
 [ 334.18082643]
 [-171.57141312]
 [  23.10485085]]
Using x02 and w : 2
[[ 627.40090574]
 [-569.10203384]
 [ 150.10753005]
 [ -11.37296477]]


Using x01 and w : 3
[[-1.40930565e+52]
 [ 2.27535273e+52]
 [-6.98654786e+51]
```

```
 [ 4.11517983e+51]]
Using x02 and w : 3
[[ 8.06978335e+53]

 [-2.60955605e+53]
 [ 2.36121669e+53]
 [-8.72145665e+52]]
```

Part (xii)

In [66]:

```python
def conjgrad(a,b,x):
    r = b - np.dot(a,x)
    p = r
    rsold = np.dot(np.transpose(r),r)

    for i in range(len(b)):
        Ap = np.dot(a,p)
        alpha = rsold/np.dot(np.transpose(p),Ap)
        x = x + np.dot(alpha, p)
        r = r - np.dot(alpha, Ap)
        rsnew = np.dot(np.transpose(r), r)
        if np.sqrt(rsnew) < 1e-8:
            break
        p = r + (rsnew/rsold)*p
        rsold = rsnew
    return x

A = [[1,2,4,8],[1,3,9,27],[1,4,16,64],[1,5,25,125]]

b = [1,-1,1,-1]

x01 = [0,0,0,0]

x02 = [1,1,1,1]

print(conjgrad(A,b,x01))
print(conjgrad(A,b,x02))
```

```
[ 9.43785229 -8.2557267  11.73874904 -3.9945521 ]
[-1.31053647 -4.20525792 -0.48505501  0.9337456 ]
```

Part (xiii)

In [67]:
```
#Solving Ax=b using x = x + e(A'Ax-A'b)
def solver(A, b, x, e):
    for k in range(100):
        x = x + e*((np.transpose(A)@A)@x - np.transpose(A)@b)
    return x

print(solver(A,b,x01,1e-6))
print(solver(A,b,x02,1e-6))
print("\n")
print(solver(A,b,x01,1e-10))
print(solver(A,b,x02,1e-10))
```

```
[0.00021242 0.00112507 0.0055775  0.02734895]
[ 1.09995546  1.43350536  2.95238442 10.02676617]


[9.25913171e-13 2.00040337e-08 1.40018220e-07 8.00084404e-07]
[1.00000296 1.00001272 1.00005695 1.00026223]
```

Part (xiv)

In [68]:
```
Q,R = np.linalg.qr(A)
print('Q = ')
print(Q)
print('R = ')
print(R)
```

```
Q =
[[-0.5         0.67082039  0.5         0.2236068 ]
 [-0.5         0.2236068  -0.5        -0.67082039]
 [-0.5        -0.2236068  -0.5         0.67082039]
 [-0.5        -0.67082039  0.5        -0.2236068 ]]
R =
[[  -2.           -7.          -27.         -112.        ]
 [   0.           -2.23606798  -15.65247584  -86.75943753]
 [   0.            0.            2.           21.        ]
 [   0.            0.            0.           -1.34164079]]
```

Part (xv)

In [69]:
```
Y = A
for i in range(1,10):
    q, r = np.linalg.qr(Y)
    Y = np.dot(r,q)

display(sp.Matrix(np.diag(Y)))
```

$$\begin{bmatrix} 139.411809943259 \\ 5.00404033727443 \\ 0.553046962185864 \\ 0.0311027572804307 \end{bmatrix}$$

Part (xvi)

In [70]:
```
A1 = np.array([[3,1],
               [1,2]])
A2 = np.array([[5,-1],
               [4,1]])
def power_method(A, y, num_iterations=10):
    for i in range(num_iterations):
        q = np.dot(A, y)
        y = q/np.linalg.norm(q)
    e = (A@y)[0]/y[0]
    return y, e[0]

for i in range(1,6):
    print("In A1 and iteration :",i)
    print(power_method(A1,i))
    print("In A2 and iteration :",i)
    print(power_method(A2,i))
    print("\n")
```

```
In A1 and iteration : 1
(array([[0.72362507, 0.44718403],
       [0.44718403, 0.27644104]]), 3.6179775280898876)
In A2 and iteration : 1
(array([[ 0.45835289, -0.19928386],
       [ 0.79713546, -0.33878257]]), 3.2608695652173902)


In A1 and iteration : 2
(array([[0.72362507, 0.44718403],
       [0.44718403, 0.27644104]]), 3.6179775280898876)
In A2 and iteration : 2
(array([[ 0.45835289, -0.19928386],
       [ 0.79713546, -0.33878257]]), 3.2608695652173902)
```

```
In A1 and iteration : 3
(array([[0.72362507, 0.44718403],
       [0.44718403, 0.27644104]]), 3.6179775280898876)
In A2 and iteration : 3
(array([[ 0.45835289, -0.19928386],
       [ 0.79713546, -0.33878257]]), 3.2608695652173902)


In A1 and iteration : 4
(array([[0.72362507, 0.44718403],
       [0.44718403, 0.27644104]]), 3.6179775280898876)
In A2 and iteration : 4
(array([[ 0.45835289, -0.19928386],
       [ 0.79713546, -0.33878257]]), 3.2608695652173902)


In A1 and iteration : 5
(array([[0.72362507, 0.44718403],
       [0.44718403, 0.27644104]]), 3.6179775280898876)
In A2 and iteration : 5
(array([[ 0.45835289, -0.19928386],
       [ 0.79713546, -0.33878257]]), 3.2608695652173902)
```

Part (xvii)

In [71]:
```
Cholesky1 = np.linalg.cholesky(A@np.transpose(A))
Cholesky2 = np.linalg.cholesky(np.transpose(A) @ A)
print('Cholesky of A*A^transpose:')
print(Cholesky1)
print('Cholesky of A^transpose*A:')
print(Cholesky2)
```

```
Cholesky of A*A^transpose:
[[  9.21954446   0.           0.           0.        ]
 [ 28.09249429   5.5508346    0.           0.        ]
 [ 63.45215891  18.46039302   1.42738182   0.        ]
 [120.50486932  41.56449685   5.19057662   0.1642757 ]]
Cholesky of A^transpose*A:
[[  2.           0.           0.           0.        ]
 [  7.           2.23606798   0.           0.        ]
 [ 27.          15.65247584   2.           0.        ]
 [112.          86.75943753  21.           1.34164079]]
```

Problem 3

In [72]:
```
A = np.array([[1,2,4,8],[1,-3,9,-27],[1,4,16,64],[1,-5,25,-125]])
```

In [73]:
```
transposed_A = np.transpose(A)
print(transposed_A)
```

```
[[   1    1    1    1]
 [   2   -3    4   -5]
 [   4    9   16   25]
 [   8  -27   64 -125]]
```

In [74]:
```
inversed_A = np.linalg.inv(A)
print('transposed_A: \n', transposed_A)
transposed_A_times_A = np.dot(transposed_A,A)
print('\n transposed_A_times_A: \n', transposed_A_times_A)
w,v = np.linalg.eig(transposed_A_times_A)
# Eigenvalues
print('\n Eigenvalue: \n', w)
singular = np.sqrt(w)
print('\n Singularvalue: \n', singular)
```

```
transposed_A:
 [[   1    1    1    1]
 [   2   -3    4   -5]
 [   4    9   16   25]
 [   8  -27   64 -125]]

 transposed_A_times_A:
 [[    4    -2    54   -80]
 [   -2    54   -80   978]
 [   54   -80   978 -2312]
 [  -80   978 -2312 20514]]

 Eigenvalue:
 [2.08305297e+04 7.12532504e+02 8.62745285e-01 6.07504615e+00]

 Singularvalue:
 [144.32785492  26.69330448   0.92884083   2.46476087]
```

Part (i)

for Rank(X) <= 3, min||A - X|| = ||A - A3|| = smallest sigma = 0.92884083 then we can know that the answer is smallest sigma, which is 0.92884083

Part (ii)

for Rank(X) <= 1, min||A - X|| = ||A - A3|| = 2nd sigma = 26.69330448 then we can know

for Rank(A) <= 1, min||A - Ak|| = ||A - As|| = 2nd sigma = [illegible] then we can know
that the answer is 2nd sigma, which is 26.69330448