different types of matrices

Np.diag((1,2))-->diag matrix np.identity((2/3)) ..>2,3 indicates order of matrix 2*2 A=A^T -->sym transpose.np.transpose(a) Np.tril(A) >>lower tri matrix np.triu(A) >>upper tri matrix

In [20]:
```
#write a program to generate identity,diagonal, symmetric and others
import numpy as np
a =np.diag((1,2,3))
a
```

Out[20]:
```
array([[1, 0, 0],
       [0, 2, 0],
       [0, 0, 3]])
```

In [21]:
```
b= np.identity((3))
b
```

Out[21]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

In [23]:
```
c=np.array([[1,2],[2,1]])
d =c.transpose()
if c.all()==d.all(): #checking each and every elements and elements of both matrices are equal
    print('true')
else:
    print('false')
```

```
true
```

In [24]:
```
np.tril(a) #applying for sq matrices only
```

Out[24]:
```
array([[1, 0, 0],
       [0, 2, 0],
       [0, 0, 3]])
```

In [25]:
```
a=np.array([[1,2,3],[2,3,4],[4,6,7]])
```

Out[25]:
```
array([[1, 0, 0],
       [2, 3, 0],
       [4, 6, 7]])
```

In [26]:
```
np.tril(a)
```

Out[26]:
```
array([[1, 0, 0],
       [2, 3, 0],
       [4, 6, 7]])
```

In [27]:
```
np.triu(a)
```

Out[27]:
```
array([[1, 2, 3],
       [0, 3, 4],
       [0, 0, 7]])
```

Determinant

np.lingalg.det(a)

In [29]:
```
a=np.array([[1,1,2],[2,1,2],[3,1,1]])
np.linalg.det(a)
```

Out[29]:
```
1.0000000000000002
```

In [30]:
```
#inverse of a matrix

a=np.array([[1,1],[2,3]])
np.linalg.inv(a)
```

Out[30]:
```
array([[ 3., -1.],
       [-2.,  1.]])
```

In [33]:
```
a=np.array([[1,2,3],[2,3,4],[4,6,7]])
a
np.linalg.inv(a)
```

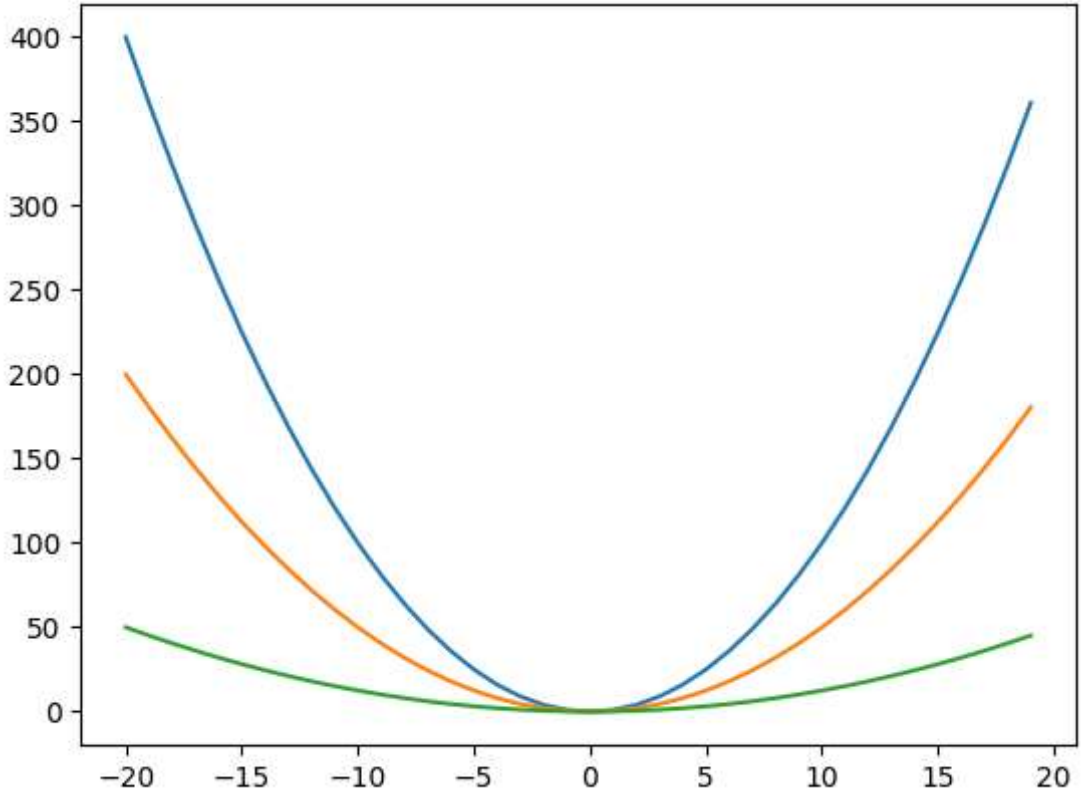
Out[33]:
```
array([[-3.,  4., -1.],
       [ 2., -5.,  2.],
       [ 0.,  2., -1.]])
```

Linear equations

In [36]:
```
import matplotlib.pyplot as mp
x=np.arange(-20,20)
y=x**2
print(y)
mp.plot(x,y) #y=x^2
mp.plot(x,y/2) #y=x^2/2
mp.plot(x,y/8)
```

```
[400 361 324 289 256 225 196 169 144 121 100  81  64  49  36  25  16   9
   4   1   0   1   4   9  16  25  36  49  64  81 100 121 144 169 196 225
 256 289 324 361]
```

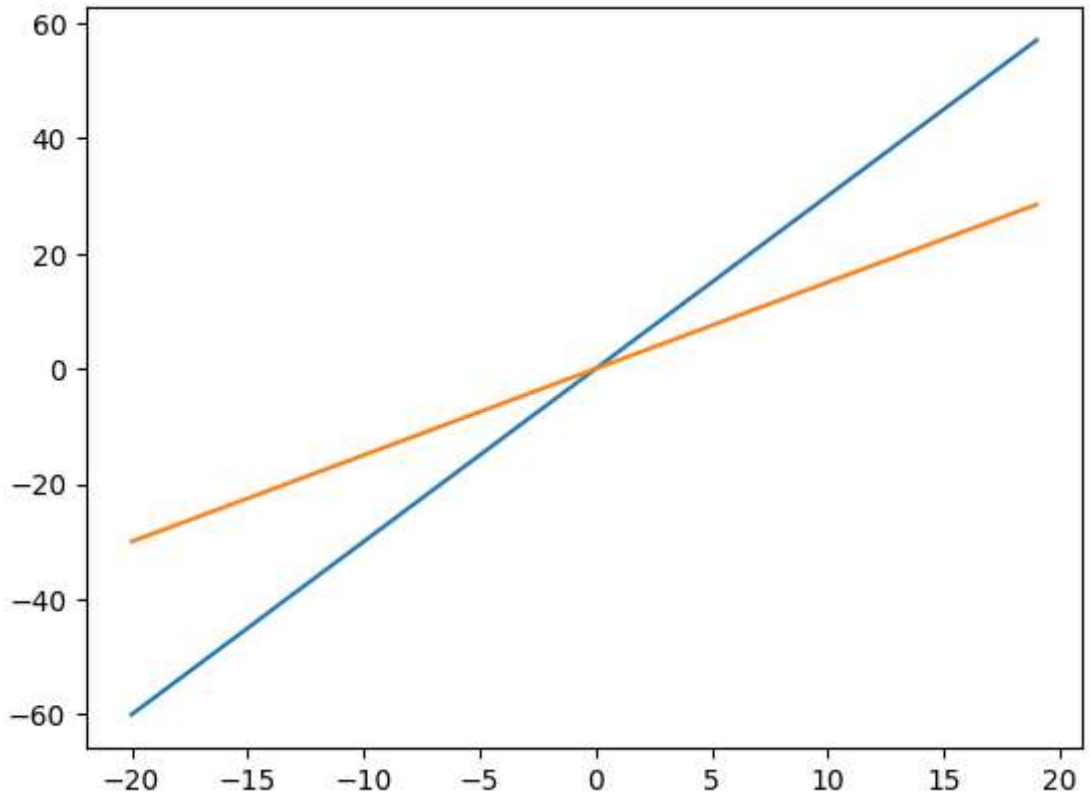
Out[36]: [<matplotlib.lines.Line2D at 0x1b762f83850>]



In [40]:
```
x=np.arange(-20,20)
y= 3*x
print(y)
mp.plot(x,y)
mp.plot(x,y/2)
```

```
[-60 -57 -54 -51 -48 -45 -42 -39 -36 -33 -30 -27 -24 -21 -18 -15 -12  -9
  -6  -3   0   3   6   9  12  15  18  21  24  27  30  33  36  39  42  45
  48  51  54  57]
```

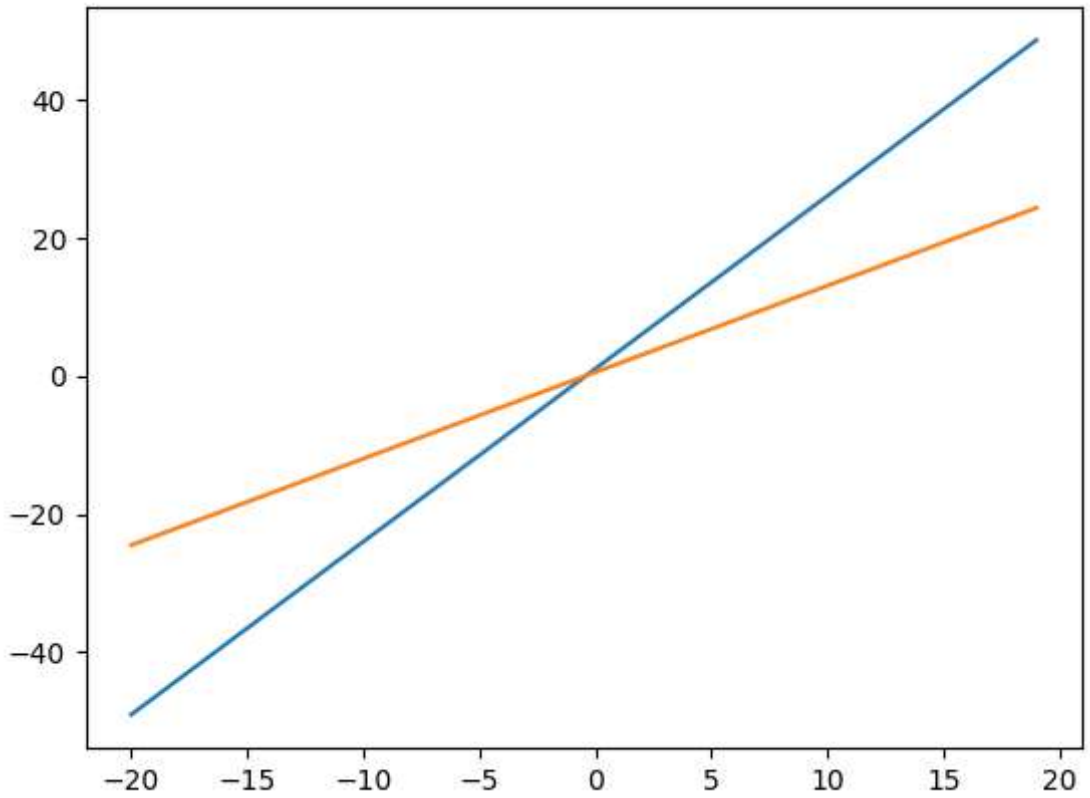
Out[40]: [<matplotlib.lines.Line2D at 0x1b76308be20>]



In [47]:
```
x=np.arange(-20,20)
y2=(2+(5*x))/2
print(y2)
mp.plot(x,y2)
mp.plot(x,y2/2)
```

```
[-49.  -46.5 -44.  -41.5 -39.  -36.5 -34.  -31.5 -29.  -26.5 -24.  -21.5
 -19.  -16.5 -14.  -11.5  -9.   -6.5  -4.   -1.5   1.    3.5   6.    8.5
  11.   13.5  16.   18.5  21.   23.5  26.   28.5  31.   33.5  36.   38.5
  41.   43.5  46.   48.5]
```

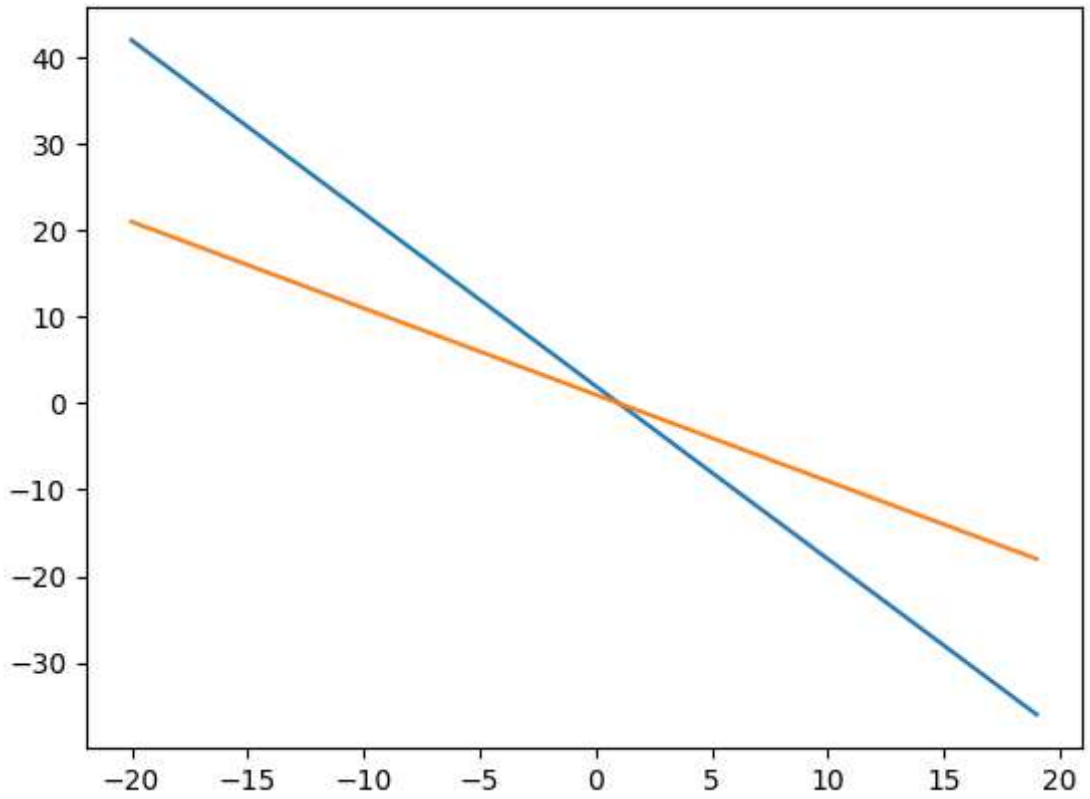
Out[47]: [<matplotlib.lines.Line2D at 0x1b764321030>]



In [48]:
```
b=np.arange(-20,20)
c= (4-4*b)/2
print(c)
mp.plot(b,c)
mp.plot(b,c/2)
```

```
[ 42.  40.  38.  36.  34.  32.  30.  28.  26.  24.  22.  20.  18.  16.
  14.  12.  10.   8.   6.   4.   2.   0.  -2.  -4.  -6.  -8. -10. -12.
 -14. -16. -18. -20. -22. -24. -26. -28. -30. -32. -34. -36.]
```

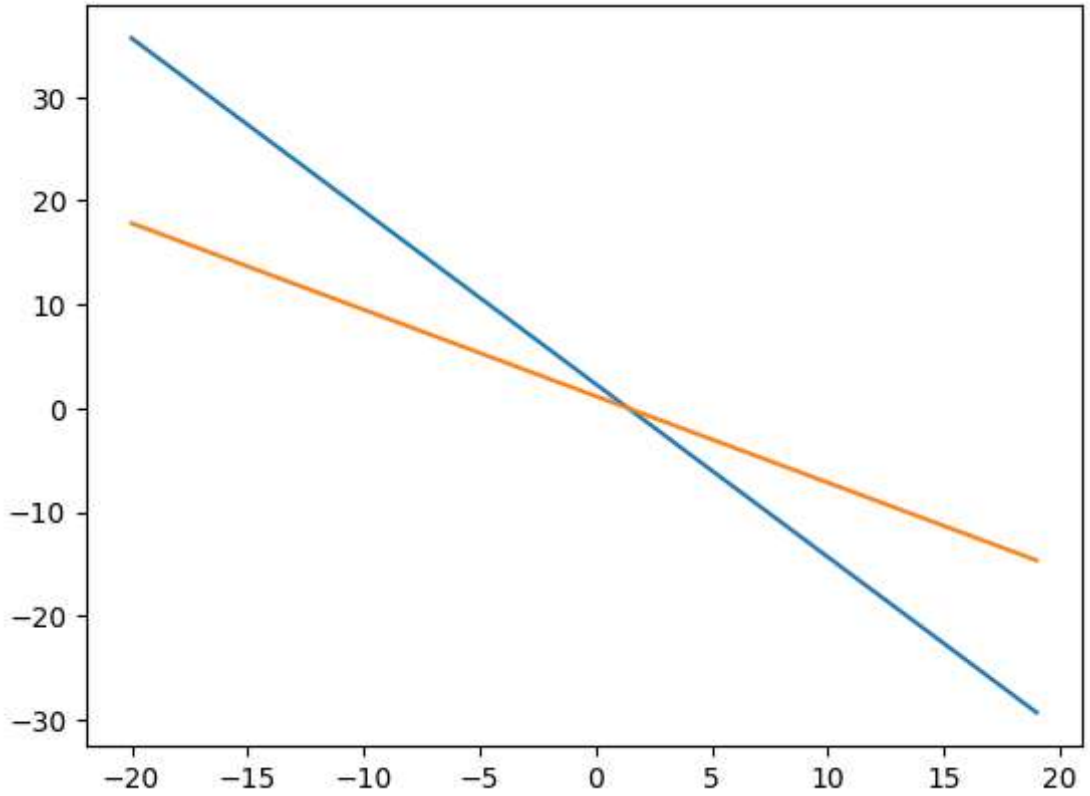
Out[48]: [<matplotlib.lines.Line2D at 0x1b76438f310>]



In [50]:
```
b1=np.arange(-20,20)
c1= (7-5*b1)/3
print(c1)
mp.plot(b1,c1)
mp.plot(b1,c1/2)
```

```
[ 35.66666667  34.          32.33333333  30.66666667  29.
  27.33333333  25.66666667  24.          22.33333333  20.66666667
  19.          17.33333333  15.66666667  14.          12.33333333
  10.66666667   9.           7.33333333   5.66666667   4.
   2.33333333   0.66666667  -1.          -2.66666667  -4.33333333
  -6.          -7.66666667  -9.33333333 -11.         -12.66666667
 -14.33333333 -16.         -17.66666667 -19.33333333 -21.
 -22.66666667 -24.33333333 -26.         -27.66666667 -29.33333333]
```

Out[50]: [<matplotlib.lines.Line2D at 0x1b7644a73a0>]



In [2]:
```
import numpy as np
```

In [4]:
```
a=np.array([[2,1],[3,4]])
x,y = np.linalg.eig(a)
```

In [5]:
```
x
```

Out[5]:
```
array([1., 5.])
```

In [6]:
```
y
```

Out[6]:
```
array([[-0.70710678, -0.31622777],
       [ 0.70710678, -0.9486833 ]])
```

Linear transformation

lineaer transformation >>input vector transformed to output vector

input and output should start at the same point

enlarge/diminishes/rotates the vector on applying transformation on vector>> scaling

rotation can't be applied on matrices due to eigen values..> only scaling is done

In [24]:
```
import numpy as np
```

In [25]:
```
x=np.array([1,2])
s=3 #scalar value
b=x*s
b
```

Out[25]:
```
array([3, 6])
```

Eigen values and eigen vectors

In [27]:
```
a=np.array([[2,1],
            [3,4]])
x,y=np.linalg.eig(a)
```

In [28]:
```
x
#displays Eigen Values
```

Out[28]:
```
array([1., 5.])
```

In [29]:
```
y
#Displays Eigen Vectors
```

Out[29]:
```
array([[-0.70710678, -0.31622777],
       [ 0.70710678, -0.9486833 ]])
```

Linear Transformation

In [30]:
```
#Consider a vector x
x=np.array([1,2])
s=3 #here s is a scalar
b=x*s
b
```

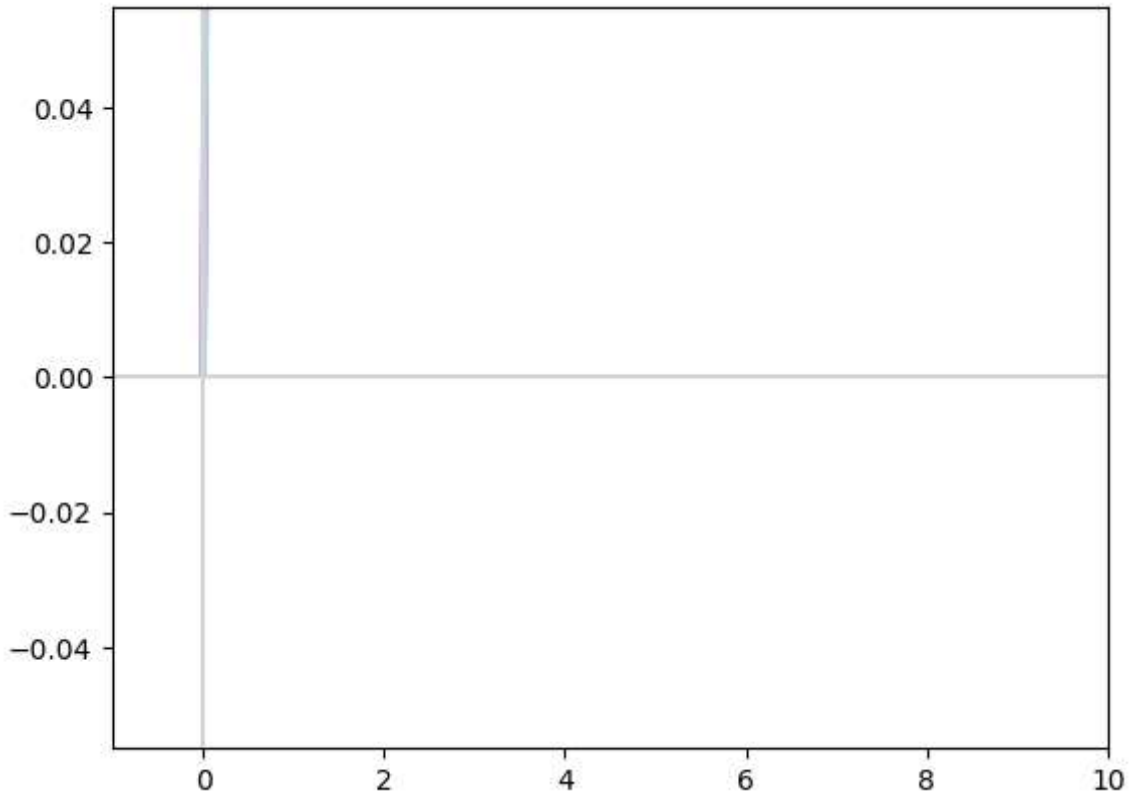
Out[30]:
```
array([3, 6])
```

In [31]:
```
import matplotlib.pyplot as plt
```

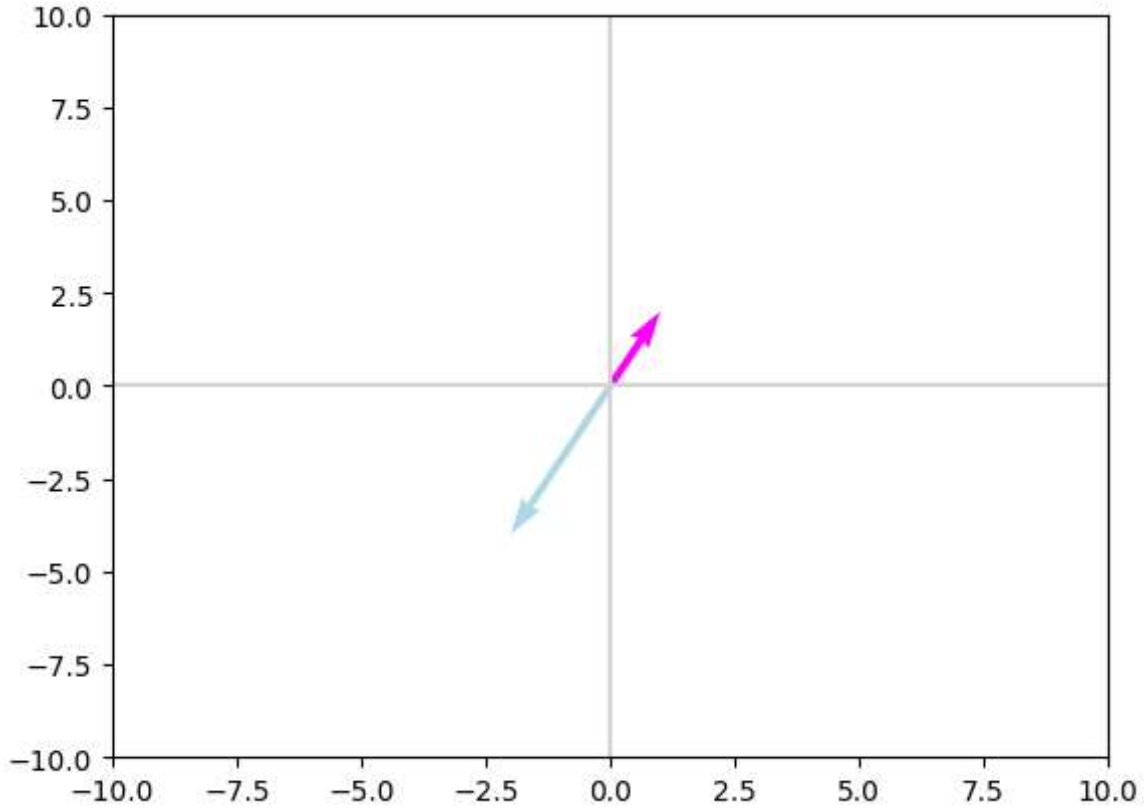
In [32]:
```
def plot_vectors(vectors,colors):
#Plot one or more vectors in a 2-D plane, specifying
    plt.axvline(x=0,color='lightgray')
    plt.axhline(y=0,color='lightgray')
    for i in range(len(vectors)):
        x=np.concatenate([[0,0],vectors[i]])
        plt.quiver([x[0]],[x[1]],[x[2]],[x[3]],
               angles='xy', scale_units='xy', scale=1, color=colors[i],)
        #here [x[0]] and [x[1]] indicate the starting point of a vector,
        # [x[2]] and [x[3] indicates the ending point of a vector
```

In [33]:
```
plot_vectors([x,b],['magenta','lightblue'])
plt.xlim(-1,10)
#_=plt.ylim(-1,10) #here x is the input vector and b is the output vector
```

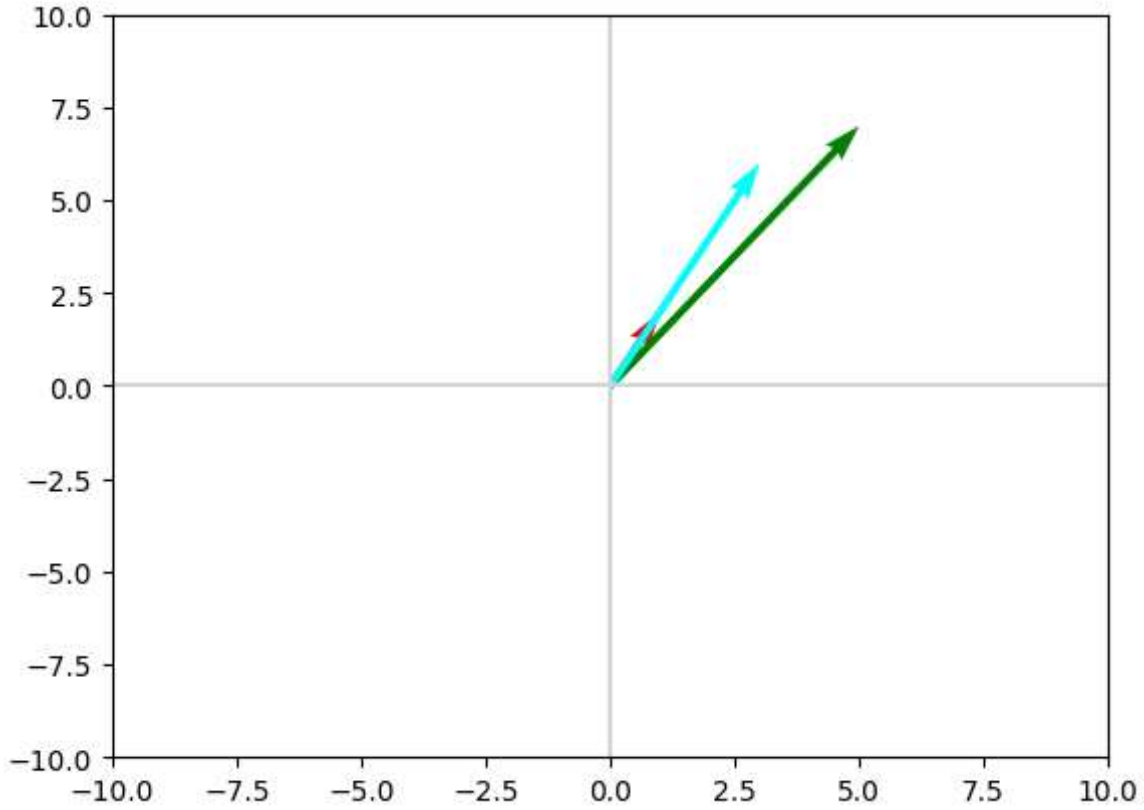
Out[33]:
```
(-1.0, 10.0)
```



In [34]:
```
# Directly printing without another scalar
s=-2
plot_vectors([x,x*s],['magenta','lightblue'])
plt.xlim(-10,10)
_=plt.ylim(-10,10)
```



In [35]:
```
#initialising another input vector as y
#x and y are the input vectors with b as the output vector
y=np.array([5,7])
plot_vectors([x,y,b],['red','green','aqua'])
plt.xlim(-10,10)
_=plt.ylim(-10,10)
```



Plotting Ellipse

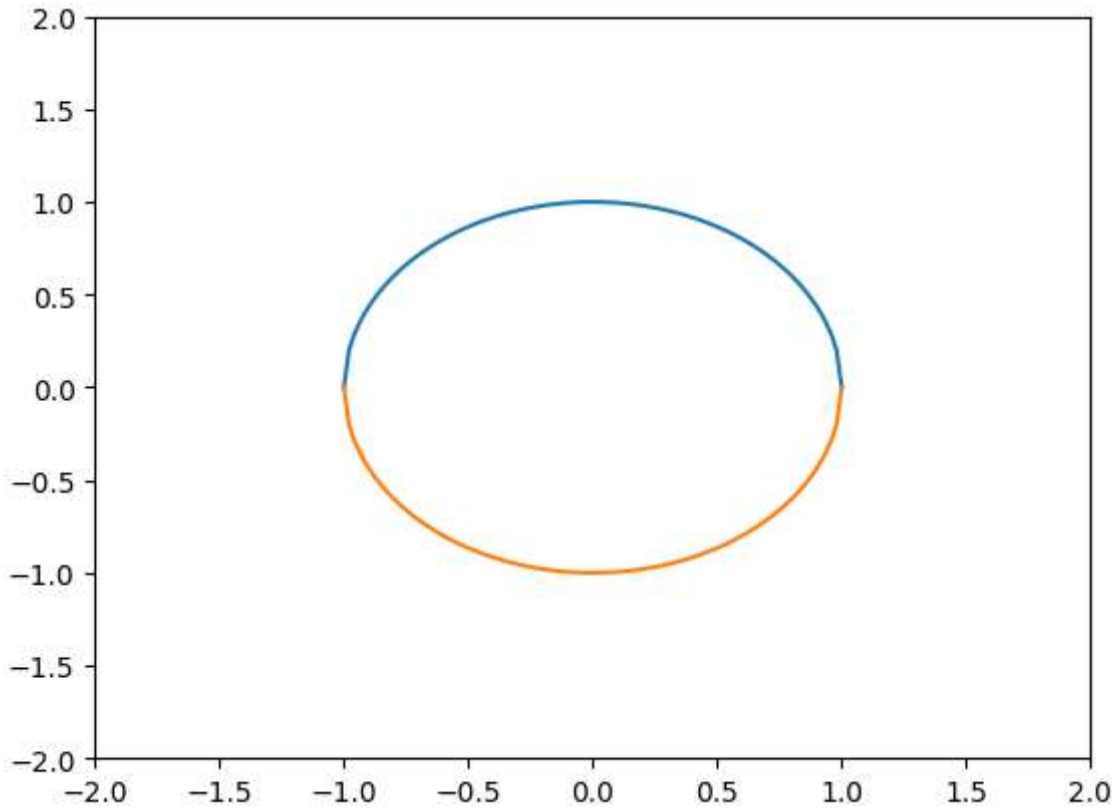
In [36]:
```
import numpy as np
x=np.linspace(-1,1,100)
#Circle equation: x^2+y^2=100
#In np.linspace(-1,1,100), we enter the co-efficients of x^2, y^2 and radius^2
#floating values between -1 to 1 are infinite we have taken 100 values in that
```

In [37]:
```
y1=np.sqrt(1-np.square(x))
```

In [38]:
```
y2= -1*y1
```

In [39]:
```
import matplotlib.pyplot as plt
plt.plot(x,y1)
```

```
plt.plot(x,y2)
plt.xlim([-2,2])
plt.ylim([-2,2])
plt.show()
```



#Transformation of an ellipse

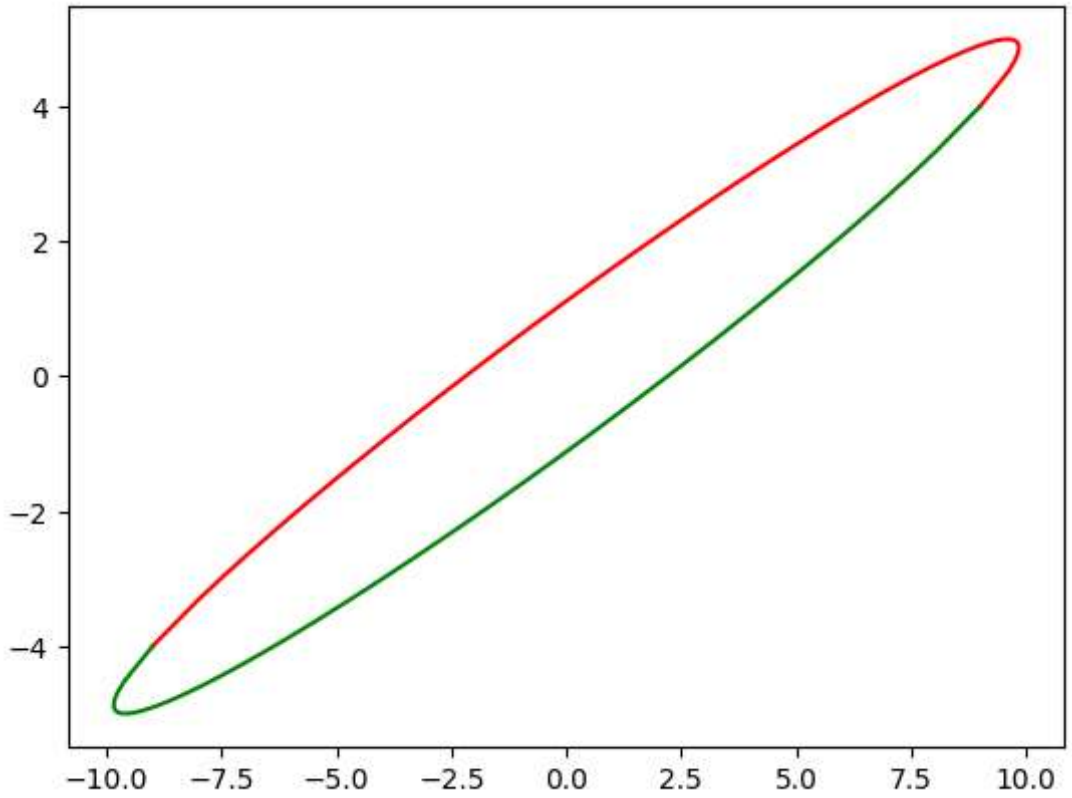
In [43]:
```
def transformation(x,y):
    return 9*x+4*y,4*x+3*y
     #[9,4],[4,3]   are the values of 2*2 matrix
```

In [46]:
```
#we take new values of transformation into x_new1,y_new1,x_new2,y_new2 variables
```

```
x_new1,y_new1=transformation(x,y1)
x_new2,y_new2=transformation(x,y2)
```

In [49]:
```
plt.plot(x_new1,y_new1,'r')
plt.plot(x_new2,y_new2,'g')
```

Out[49]: [<matplotlib.lines.Line2D at 0x2d7460a5f60>]



In [ ]:
```
##Here, circle is transformed into an ellipse
```

In [50]:
```
eigen,eigenvector=np.linalg.eig(np.array([[9,4],[4,3]]))
```

In [51]:
```
eigen
```

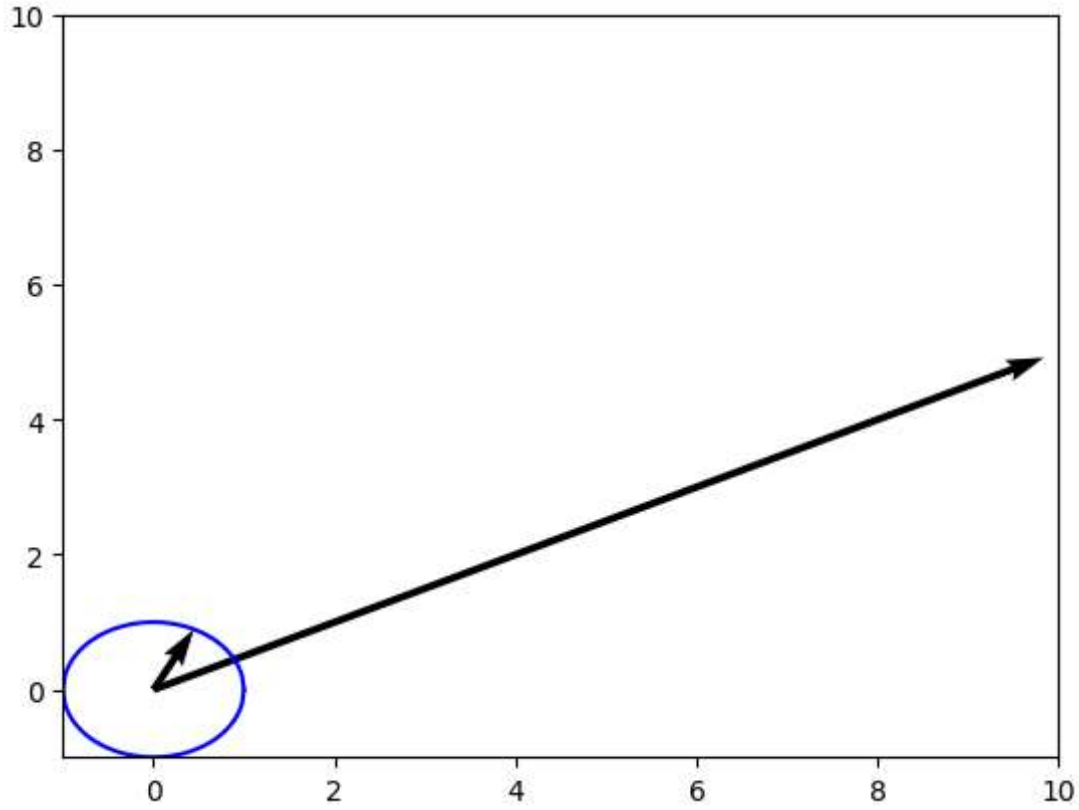
Out[51]: array([11., 1.])

In [52]:
```
eigenvector
```

Out[52]:
```
array([[ 0.89442719, -0.4472136 ],
       [ 0.4472136 ,  0.89442719]])
```

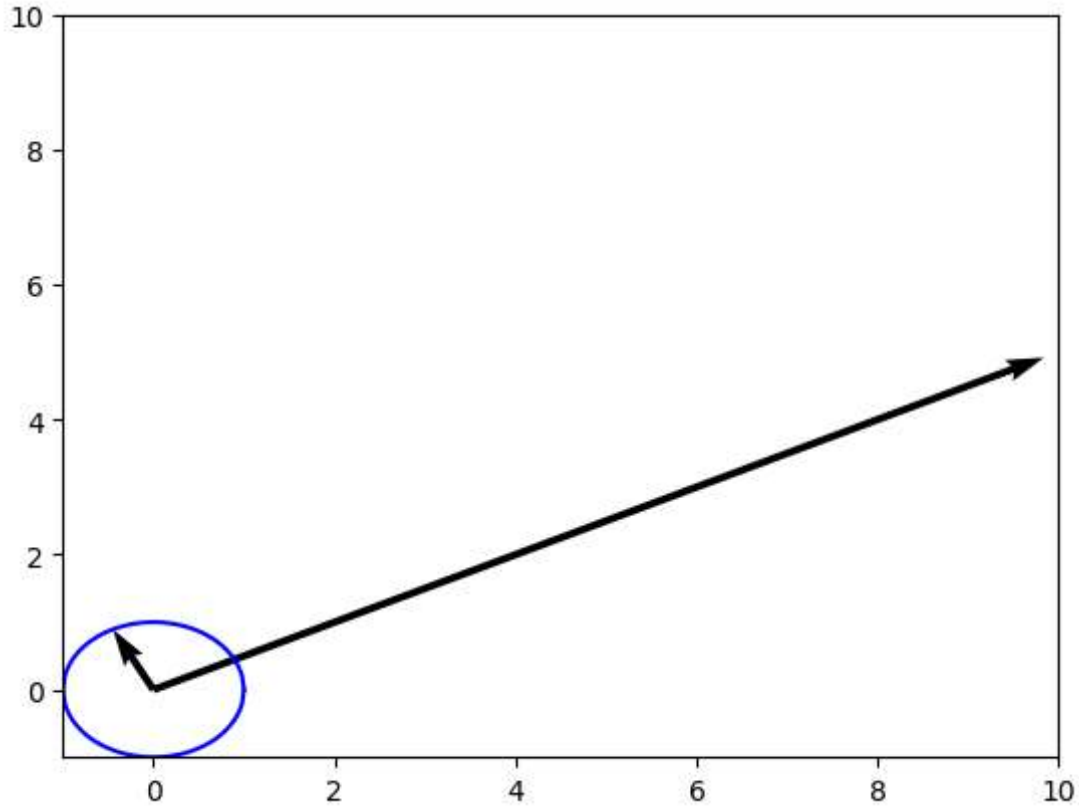
In [53]:
```
soa=np.array([[0,0,eigen[0]*eigenvector[0][0],eigen[0]*eigenvector[1][0]]])
sol=np.array([[0,0,eigen[1]*eigenvector[1][0],eigen[1]*eigenvector[1][1]]])
#[]
X,Y,U,V=zip(*soa)
X1,Y1,U1,V1=zip(*sol)
plt.plot(x,y1,'b')
plt.plot(x,y2,'b')
plt.quiver(X,Y,U,V,
          angles='xy',scale_units='xy', scale=1)
plt.quiver(X1,Y1,U1,V1,
          angles='xy',scale_units='xy', scale=1)
plt.xlim([-1,10])
plt.ylim([-1,10])
```

Out[53]:
```
(-1.0, 10.0)
```



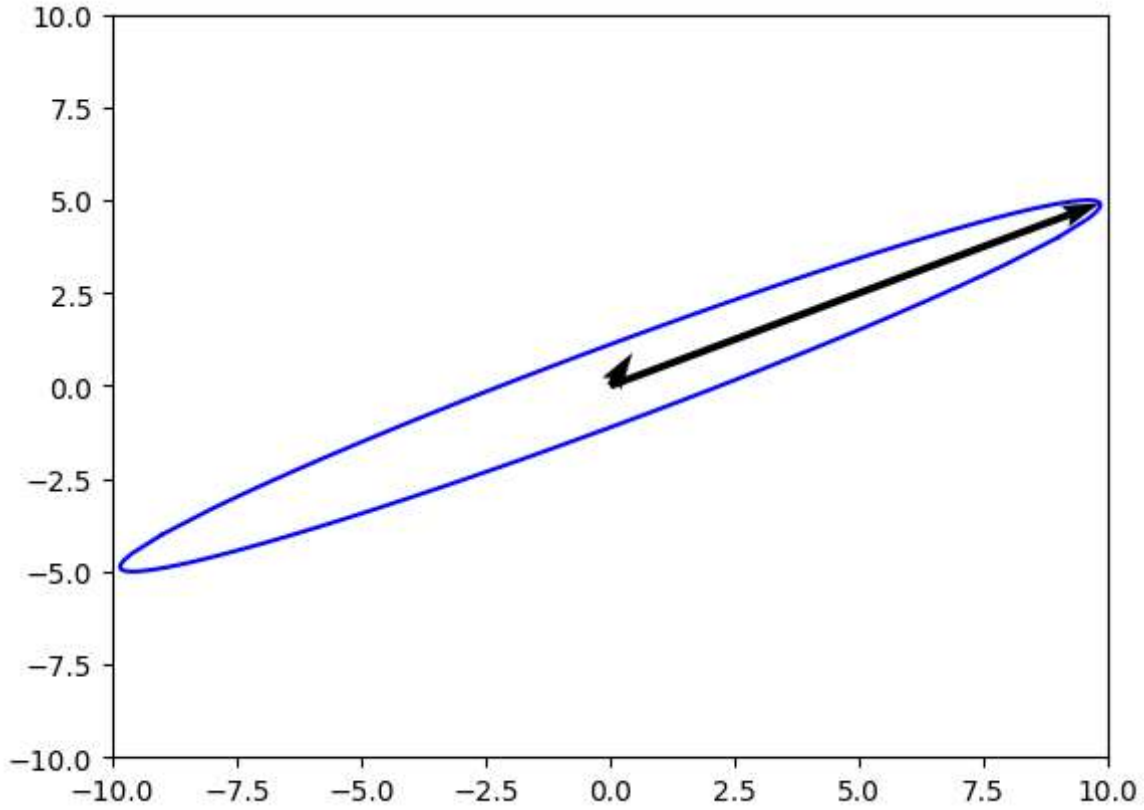
In [54]:
```
#Changed the index from eigen[1]*eigenvector[1][0] toeigen[1]*eigenvector[0][1]
#Column Change
soa=np.array([[0,0,eigen[0]*eigenvector[0][0],eigen[0]*eigenvector[1][0]]])
sol=np.array([[0,0,eigen[1]*eigenvector[0][1],eigen[1]*eigenvector[1][1]]])
X,Y,U,V=zip(*soa)
X1,Y1,U1,V1=zip(*sol)
plt.plot(x,y1,'b')
plt.plot(x,y2,'b')
plt.quiver(X,Y,U,V,
          angles='xy',scale_units='xy', scale=1)
plt.quiver(X1,Y1,U1,V1,
          angles='xy',scale_units='xy', scale=1)
plt.xlim([-1,10])
plt.ylim([-1,10])
```

Out[54]:
```
(-1.0, 10.0)
```



In [55]:
```
#Changed the value from x,y1,x,y2 to x_new1,y_new1 and x_new2,y_new2
#Column Change
soa=np.array([[0,0,eigen[0]*eigenvector[0][0],eigen[0]*eigenvector[1][0]]])
sol=np.array([[0,0,eigen[1]*eigenvector[1][0],eigen[1]*eigenvector[1][1]]])
X,Y,U,V=zip(*soa)
X1,Y1,U1,V1=zip(*sol)
plt.plot(x_new1,y_new1,'b')
plt.plot(x_new2,y_new2,'b')
plt.quiver(X,Y,U,V,
          angles='xy',scale_units='xy', scale=1)
plt.quiver(X1,Y1,U1,V1,
          angles='xy',scale_units='xy', scale=1)
plt.xlim([-10,10])
plt.ylim([-10,10])
```

Out[55]:
```
(-10.0, 10.0)
```



Singular Value Decompostion

In [57]:
```
#svd[a] eigenvector of a and a^t d= sq root of eigen values v^t = a^t and a
#svd returns 3 values
#svd is applied to reduce the size of images
#Singular Value Decompostion (SVD): The Singlular Value Decompostion (SVD) of a matrix
#is a factorization of that the matrix into three matrices.
#U,D,VT
#U--Eigenvector of AA^T
#D--Squareroot of eigenvalues
#V^T--#right singular value, i.e.., A^TA
#Compress an image
```

In [ ]:
```
#Compressing an Image
```

In [59]:
```
A=np.array([[-1,2],[3,-2],[5,7]])
A
```

Out[59]:
```
array([[-1,  2],
       [ 3, -2],
       [ 5,  7]])
```

In [60]:
```
U,d,VT= np.linalg.svd(A)
```

In [61]:
```
U #the colomn values of u called left singular vector
```

Out[61]:
```
array([[ 0.12708324,  0.47409506,  0.87125411],
       [ 0.00164602, -0.87847553,  0.47778451],
       [ 0.99189069, -0.0592843 , -0.11241989]])
```

In [62]:
```
VT# >>> #Column value of right singular vector
```

Out[62]:
```
array([[ 0.55798885,  0.82984845],
       [-0.82984845,  0.55798885]])
```

In [64]:
```
d #diag values of singular value
```

Out[64]:
```
array([8.66918448, 4.10429538])
```

In [65]:
```
from PIL import Image
```

In [66]:
```
img = Image.open('modi1.jpg.jpeg')
_=plt.imshow(img)
```

In [67]:
```
imggray = img.convert('LA') #LA indicates color band
_=plt.imshow(imggray)
```

In [69]:
```
#creating new matrix from image before applying svd
#band=1 rgb
#band =0 for red 1 for green 2 for blue
#size 0 >> no of rows present and size 1>> no.of coloumns
#svd compresses the image
imgmat=np.array(list(imggray.getdata(band=0)),float)
imgmat.shape =(imggray.size[1],imggray.size[0])
imgmat=np.matrix(imgmat)
_=plt.imshow(imgmat, cmap='gray')
```

In [70]:
```
U, sigma,V =np.linalg.svd(imgmat)
```

In [71]:
```
U
```

Out[71]:
```
matrix([[-0.04759306,  0.02120062,  0.03605262, ...,  0.00534043,
          0.01610291,  0.02993521],
        [-0.04759571,  0.02127977,  0.0359307 , ...,  0.0043117 ,
         -0.07943641, -0.03956248],
        [-0.04762426,  0.02141364,  0.03607913, ..., -0.00332636,
          0.05008565,  0.00585326],
        ...,
        [-0.04537899, -0.02159797,  0.08780722, ..., -0.0010821 ,
         -0.00111973, -0.00021762],
        [-0.04534663, -0.02054186,  0.08475936, ...,  0.00020093,
          0.00351981, -0.0012747 ],
        [-0.04530587, -0.01970305,  0.08261453, ...,  0.00023264,
         -0.00297974,  0.0015839 ]])
```

In [72]:
```
V
```

Out[72]:
```
matrix([[-0.03516809, -0.03520418, -0.03526966, ..., -0.03746736,
         -0.0374089 , -0.03739109],
        [-0.00572139, -0.00620153, -0.00698787, ..., -0.02780694,
         -0.02779575, -0.02773563],
        [-0.06947448, -0.06993877, -0.07049955, ..., -0.01715403,
         -0.01722642, -0.01726522],
        ...,
        [ 0.0045503 , -0.00279934, -0.01781835, ...,  0.79109661,
         -0.05618234, -0.06572806],
        [-0.05452047, -0.01220729,  0.01027495, ..., -0.04645416,
          0.77358226, -0.06230439],
        [-0.02596574,  0.00392267,  0.00405014, ..., -0.02919379,
         -0.02963976,  0.74059022]])
```

In [73]:
```
sigma
```

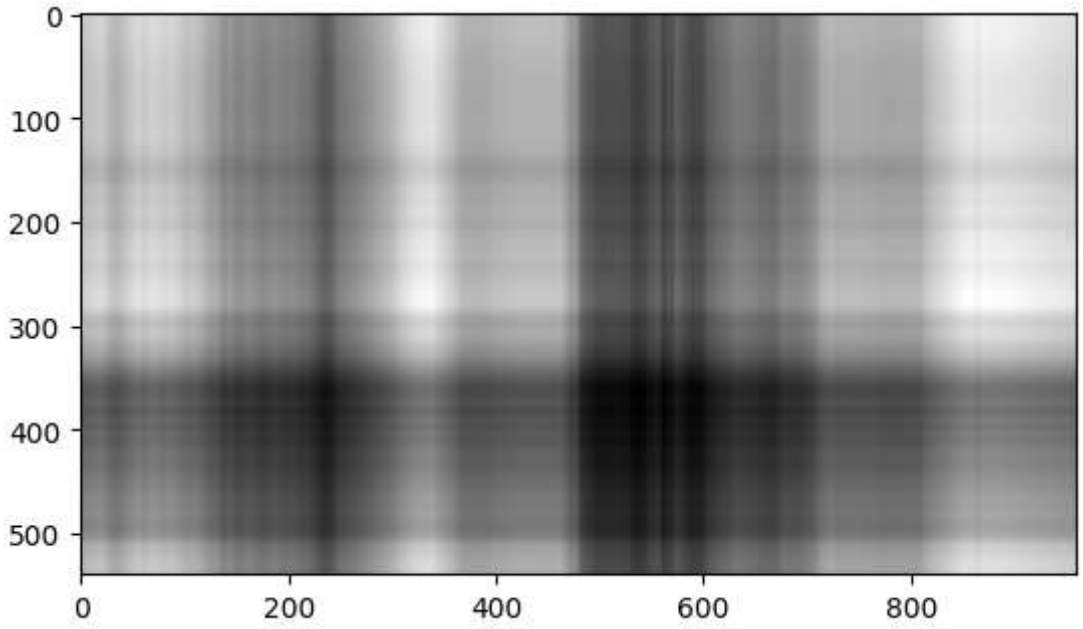
Out[73]:
```
array([1.39617756e+05, 2.07387337e+04, 1.44637774e+04, 1.24224845e+04,
       9.82499351e+03, 8.28718025e+03, 7.76586823e+03, 6.08976516e+03,
       4.94608675e+03, 4.75129425e+03, 4.10933494e+03, 3.58078398e+03,
       3.47809188e+03, 3.22403074e+03, 2.89287851e+03, 2.81016782e+03,
       2.61409305e+03, 2.39554512e+03, 2.24189338e+03, 2.20566843e+03,
       2.05941918e+03, 1.96053731e+03, 1.83318235e+03, 1.80505193e+03,
       1.77128447e+03, 1.73844682e+03, 1.72845085e+03, 1.65298952e+03,
       1.64128038e+03, 1.59099438e+03, 1.56440590e+03, 1.45847421e+03,
       1.40807315e+03, 1.38456869e+03, 1.37123507e+03, 1.33663498e+03,
       1.30466006e+03, 1.26464839e+03, 1.24778185e+03, 1.21453104e+03,
       1.19115790e+03, 1.13003035e+03, 1.10675280e+03, 1.07487462e+03,
       1.06582225e+03, 1.03942994e+03, 1.01215904e+03, 1.00116796e+03,
       9.80452346e+02, 9.66643889e+02, 9.32424096e+02, 9.16660212e+02,
       8.99665508e+02, 8.91109672e+02, 8.71135759e+02, 8.58619107e+02,
       8.40786412e+02, 8.15021637e+02, 8.01947255e+02, 7.93812423e+02,
       7.85572827e+02, 7.71834543e+02, 7.51345953e+02, 7.36410462e+02,
       7.22327452e+02, 7.05894336e+02, 6.95608000e+02, 6.87287913e+02,
       6.72895643e+02, 6.66884072e+02, 6.59436876e+02, 6.50667686e+02,
       6.38679033e+02, 6.31872559e+02, 6.22868440e+02, 6.18990981e+02,
       6.05889122e+02, 5.94981766e+02, 5.89062745e+02, 5.78339150e+02,
       5.76550326e+02, 5.62721493e+02, 5.52713587e+02, 5.46792842e+02,
       5.38143110e+02, 5.27274745e+02, 5.21306646e+02, 5.10066479e+02,
       4.99335552e+02, 4.92172237e+02, 4.85220202e+02, 4.82702498e+02,
       4.73340884e+02, 4.70262088e+02, 4.65801867e+02, 4.60736679e+02,
       4.55654230e+02, 4.48969391e+02, 4.39858736e+02, 4.32614360e+02,
       4.31190115e+02, 4.26344691e+02, 4.24072596e+02, 4.19508949e+02,
       4.15499671e+02, 4.14493061e+02, 4.02745066e+02, 3.96726690e+02,
       3.95013151e+02, 3.83742045e+02, 3.80573671e+02, 3.74135952e+02,
       3.71875553e+02, 3.67646603e+02, 3.61693870e+02, 3.60192179e+02,
       3.55511264e+02, 3.50616938e+02, 3.48528246e+02, 3.44896074e+02,
       3.37230889e+02, 3.32359399e+02, 3.29465486e+02, 3.25876588e+02,
       3.22007403e+02, 3.13370464e+02, 3.11358627e+02, 3.08206982e+02,
       3.06787668e+02, 3.00826238e+02, 2.98333153e+02, 2.95170125e+02,
       2.89219456e+02, 2.83708609e+02, 2.83105302e+02, 2.79024187e+02,
       2.77365928e+02, 2.71336781e+02, 2.68125908e+02, 2.66296478e+02,
       2.63477188e+02, 2.62610369e+02, 2.60696257e+02, 2.54036979e+02,
       2.53286823e+02, 2.50032906e+02, 2.49467706e+02, 2.42825307e+02,
       2.39999737e+02, 2.39312929e+02, 2.38453249e+02, 2.32717778e+02,
       2.30501222e+02, 2.28945834e+02, 2.26330860e+02, 2.25207721e+02,
       2.20158967e+02, 2.18326615e+02, 2.16992411e+02, 2.14055607e+02,
       2.11590565e+02, 2.08849298e+02, 2.05953966e+02, 2.03506997e+02,
```

```
2.01758339e+02, 1.98698340e+02, 1.97216008e+02, 1.95529649e+02,
1.93038949e+02, 1.92165327e+02, 1.88114325e+02, 1.86194311e+02,
1.84357448e+02, 1.82920224e+02, 1.80795354e+02, 1.78643017e+02,
1.75844585e+02, 1.74643346e+02, 1.72849228e+02, 1.70613544e+02,
1.68607441e+02, 1.67886122e+02, 1.65935778e+02, 1.65086822e+02,
1.62861204e+02, 1.60088549e+02, 1.59100436e+02, 1.57839801e+02,
1.55937104e+02, 1.54119190e+02, 1.50981350e+02, 1.48934362e+02,
1.47963749e+02, 1.46786879e+02, 1.46310830e+02, 1.44347183e+02,
1.41652282e+02, 1.40741289e+02, 1.39211929e+02, 1.36824453e+02,
1.36231760e+02, 1.32861676e+02, 1.32104516e+02, 1.31753565e+02,
1.30503766e+02, 1.28387171e+02, 1.27504790e+02, 1.25103192e+02,
1.23702230e+02, 1.22720477e+02, 1.21263588e+02, 1.20889059e+02,
1.19388731e+02, 1.18762613e+02, 1.17769038e+02, 1.17029666e+02,
1.15544455e+02, 1.15087995e+02, 1.13888953e+02, 1.13039179e+02,
1.10666190e+02, 1.09273302e+02, 1.07503440e+02, 1.05381385e+02,
1.05076847e+02, 1.03804871e+02, 1.02524330e+02, 1.01993759e+02,
1.01047369e+02, 1.00253071e+02, 9.87336452e+01, 9.77332271e+01,
9.69586708e+01, 9.51982777e+01, 9.47926355e+01, 9.38961961e+01,
9.34513865e+01, 9.10387737e+01, 9.00486196e+01, 8.98325125e+01,
8.84968948e+01, 8.78486502e+01, 8.66611773e+01, 8.52173871e+01,
8.41817112e+01, 8.30023601e+01, 8.25708468e+01, 8.17537560e+01,
7.97115544e+01, 7.93578659e+01, 7.77604624e+01, 7.74957194e+01,
7.70924348e+01, 7.65120123e+01, 7.54624783e+01, 7.47813965e+01,
7.42823104e+01, 7.35339929e+01, 7.29361008e+01, 7.16997392e+01,
7.14609931e+01, 6.93177787e+01, 6.83334099e+01, 6.75578632e+01,
6.68086987e+01, 6.65275309e+01, 6.55924548e+01, 6.49484758e+01,
6.41543647e+01, 6.35262115e+01, 6.29201645e+01, 6.19696293e+01,
6.17528769e+01, 6.05752610e+01, 5.87800359e+01, 5.84128485e+01,
5.73965384e+01, 5.73086875e+01, 5.67400791e+01, 5.63237567e+01,
5.56475313e+01, 5.51213013e+01, 5.42039602e+01, 5.37671225e+01,
5.29712446e+01, 5.23406327e+01, 5.18306863e+01, 5.12726158e+01,
5.04620658e+01, 4.98431360e+01, 4.88974388e+01, 4.84956796e+01,
4.77281950e+01, 4.69179619e+01, 4.64124122e+01, 4.61465592e+01,
4.58367804e+01, 4.52562517e+01, 4.44436743e+01, 4.41399647e+01,
4.33919196e+01, 4.30590869e+01, 4.24065910e+01, 4.20712532e+01,
4.13080776e+01, 4.09998238e+01, 4.05418897e+01, 4.03159911e+01,
3.94701965e+01, 3.92117731e+01, 3.89530930e+01, 3.79902001e+01,
3.74793711e+01, 3.72880649e+01, 3.64378560e+01, 3.55410297e+01,
3.54811639e+01, 3.47727039e+01, 3.41764508e+01, 3.35597958e+01,
3.32290981e+01, 3.28637617e+01, 3.25192870e+01, 3.15798620e+01,
3.12758159e+01, 3.10126313e+01, 3.05530403e+01, 3.01987815e+01,
2.98656235e+01, 2.95788960e+01, 2.94643829e+01, 2.91427866e+01,
```

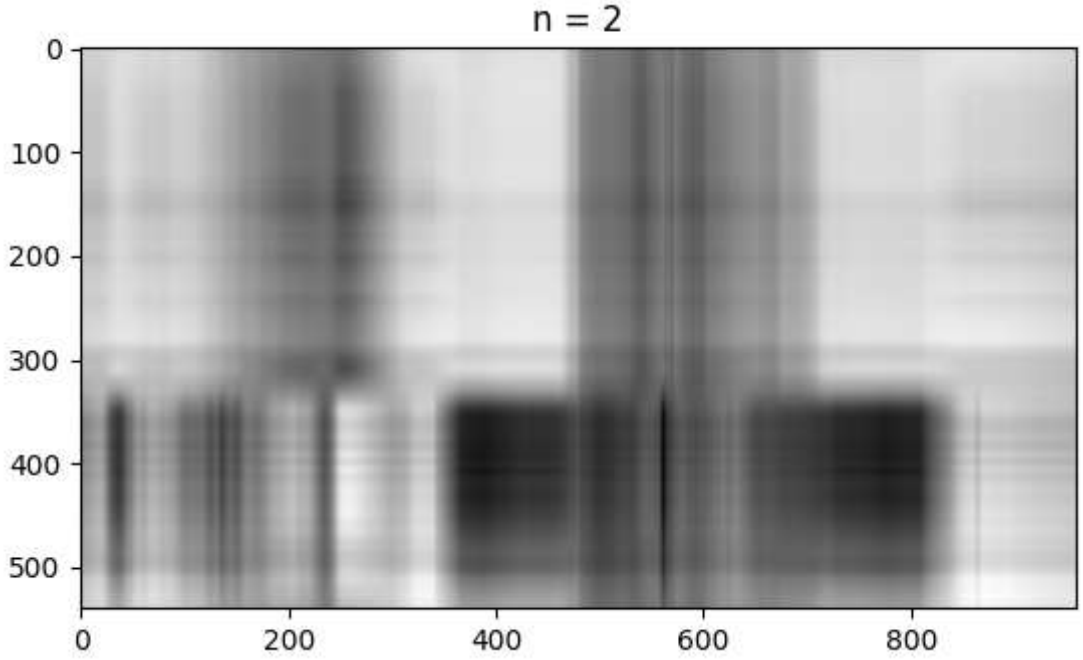
2.85264430e+01, 2.79181457e+01, 2.76815015e+01, 2.75026121e+01,
2.71136288e+01, 2.66409157e+01, 2.60516607e+01, 2.56948033e+01,
2.53926528e+01, 2.49763825e+01, 2.49017965e+01, 2.44594211e+01,
2.39602752e+01, 2.37373072e+01, 2.35924344e+01, 2.28563811e+01,
2.26116437e+01, 2.23812617e+01, 2.19073656e+01, 2.18504468e+01,
2.16573074e+01, 2.11939807e+01, 2.09100550e+01, 2.05986795e+01,
2.01317336e+01, 1.98677438e+01, 1.92814477e+01, 1.89798123e+01,
1.86172250e+01, 1.84612775e+01, 1.83853576e+01, 1.78043971e+01,
1.76338141e+01, 1.75033633e+01, 1.73545090e+01, 1.69841575e+01,
1.66978935e+01, 1.65235498e+01, 1.62154895e+01, 1.59098972e+01,
1.56457683e+01, 1.54038528e+01, 1.51579893e+01, 1.49217143e+01,
1.47831824e+01, 1.46200448e+01, 1.42330104e+01, 1.39435033e+01,
1.37449651e+01, 1.32864108e+01, 1.30696505e+01, 1.29206960e+01,
1.27436430e+01, 1.24879119e+01, 1.23675740e+01, 1.20434754e+01,
1.18874967e+01, 1.13790235e+01, 1.12199622e+01, 1.09543102e+01,
1.08872034e+01, 1.06383525e+01, 1.03967951e+01, 1.03447456e+01,
9.97110323e+00, 9.77639413e+00, 9.64065929e+00, 9.52672261e+00,
9.38666938e+00, 8.94535696e+00, 8.86518140e+00, 8.72137973e+00,
8.62498659e+00, 8.38713358e+00, 8.31920731e+00, 8.21892393e+00,
8.11877487e+00, 8.03216443e+00, 7.94549465e+00, 7.77174222e+00,
7.63731111e+00, 7.55518454e+00, 7.51451384e+00, 7.45132854e+00,
7.33234320e+00, 7.28087182e+00, 7.25950106e+00, 7.08800807e+00,
7.01535041e+00, 6.98046585e+00, 6.89333406e+00, 6.85058624e+00,
6.76763286e+00, 6.68538039e+00, 6.61399069e+00, 6.53679537e+00,
6.51538913e+00, 6.37108571e+00, 6.31766793e+00, 6.22659075e+00,
6.18905428e+00, 6.13316215e+00, 6.09802626e+00, 5.98994860e+00,
5.93048213e+00, 5.90009676e+00, 5.86589573e+00, 5.80526807e+00,
5.75433098e+00, 5.71234076e+00, 5.64956305e+00, 5.63959675e+00,
5.55007185e+00, 5.48489235e+00, 5.44928779e+00, 5.36318228e+00,
5.31860712e+00, 5.25142144e+00, 5.18556471e+00, 5.16306698e+00,
5.12615508e+00, 5.08978370e+00, 5.03577945e+00, 5.00902456e+00,
4.95325869e+00, 4.90719362e+00, 4.86361141e+00, 4.83136291e+00,
4.77058941e+00, 4.70494970e+00, 4.64995812e+00, 4.63685223e+00,
4.55629794e+00, 4.53415411e+00, 4.49815624e+00, 4.42646652e+00,
4.39860438e+00, 4.37651081e+00, 4.34783884e+00, 4.30078766e+00,
4.25928377e+00, 4.20953065e+00, 4.19064469e+00, 4.18269098e+00,
4.12090207e+00, 4.09228888e+00, 4.03377065e+00, 3.99596775e+00,
3.97225743e+00, 3.92561360e+00, 3.89308524e+00, 3.84797669e+00,
3.83245136e+00, 3.78030691e+00, 3.76793723e+00, 3.72242358e+00,
3.70198996e+00, 3.65202018e+00, 3.62464431e+00, 3.59787477e+00,
3.56834933e+00, 3.49942531e+00, 3.47002867e+00, 3.44558044e+00,
3.43142485e+00, 3.40786985e+00, 3.37427345e+00, 3.35339419e+00,

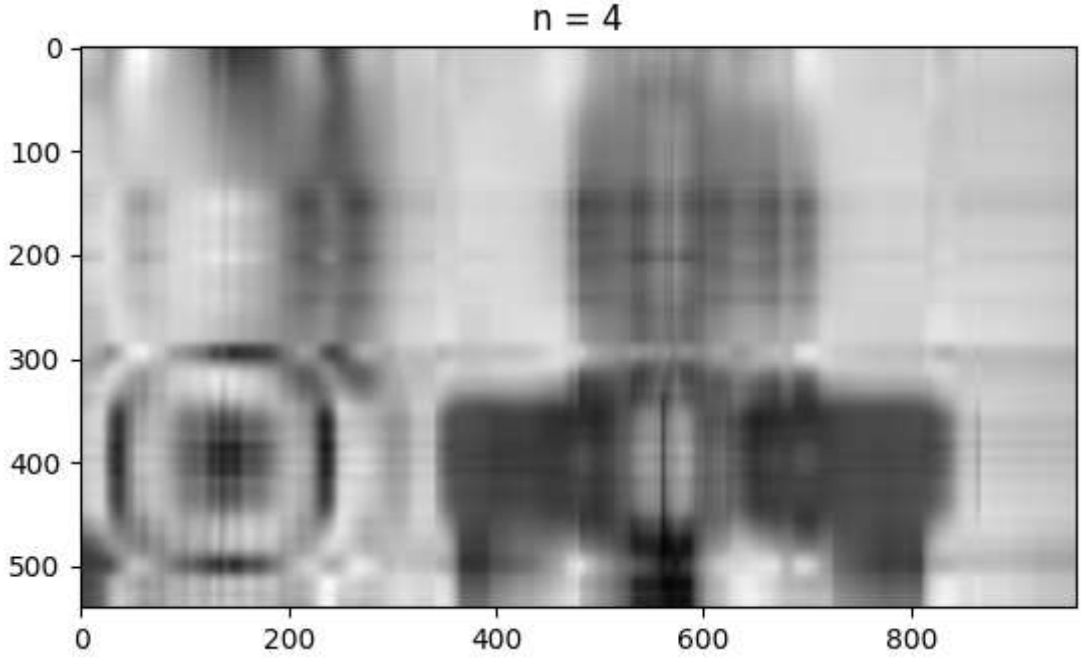
```
3.28045967e+00, 3.24834745e+00, 3.24031247e+00, 3.18259766e+00,
3.13115217e+00, 3.12054635e+00, 3.08381761e+00, 3.07016196e+00,
3.05266743e+00, 3.01029164e+00, 2.99636747e+00, 2.95196478e+00,
2.90931935e+00, 2.87704909e+00, 2.84159910e+00, 2.78846614e+00,
2.77605309e+00, 2.75453787e+00, 2.69327923e+00, 2.64400785e+00,
2.62682142e+00, 2.59098894e+00, 2.54371650e+00, 2.50859615e+00,
2.49769518e+00, 2.46344279e+00, 2.43398660e+00, 2.39819120e+00,
2.38412328e+00, 2.30976665e+00, 2.25554614e+00, 2.22455624e+00,
2.18219638e+00, 2.16038282e+00, 2.14262675e+00, 2.08305615e+00,
2.05163170e+00, 2.02275790e+00, 1.93454063e+00, 1.90288386e+00])
```

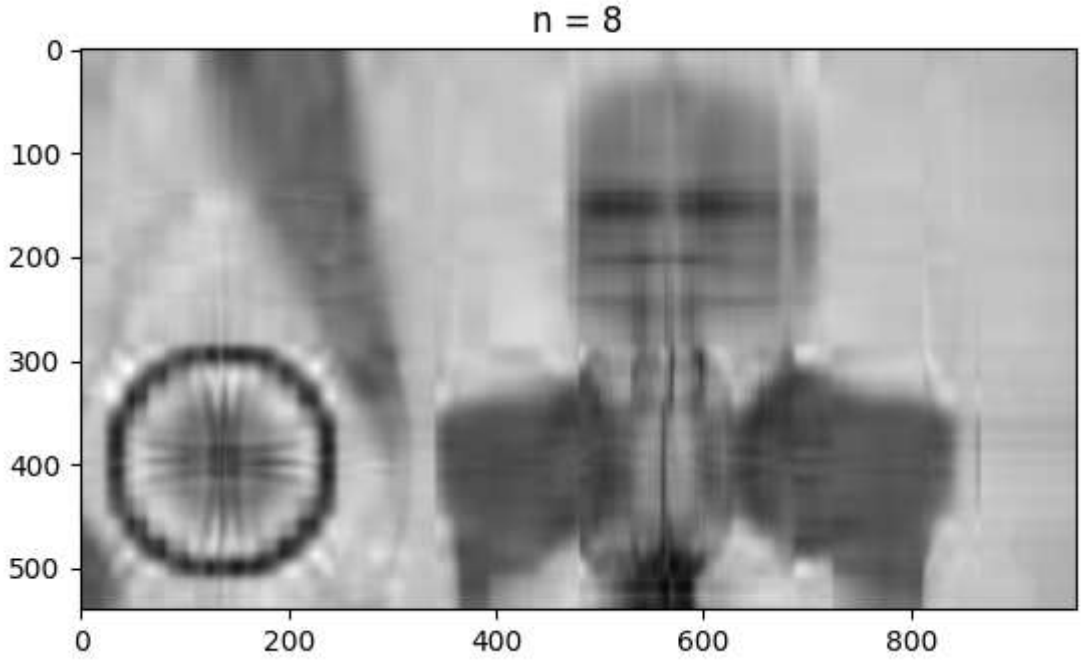
In [74]:
```
#reconstructing image
reconstimg= np.matrix(U[:,:1])*np.diag(sigma[:1])*np.matrix(V[:1, :]) #:1 means 1 coloumn
_= plt.imshow(reconstimg, cmap='gray')
```

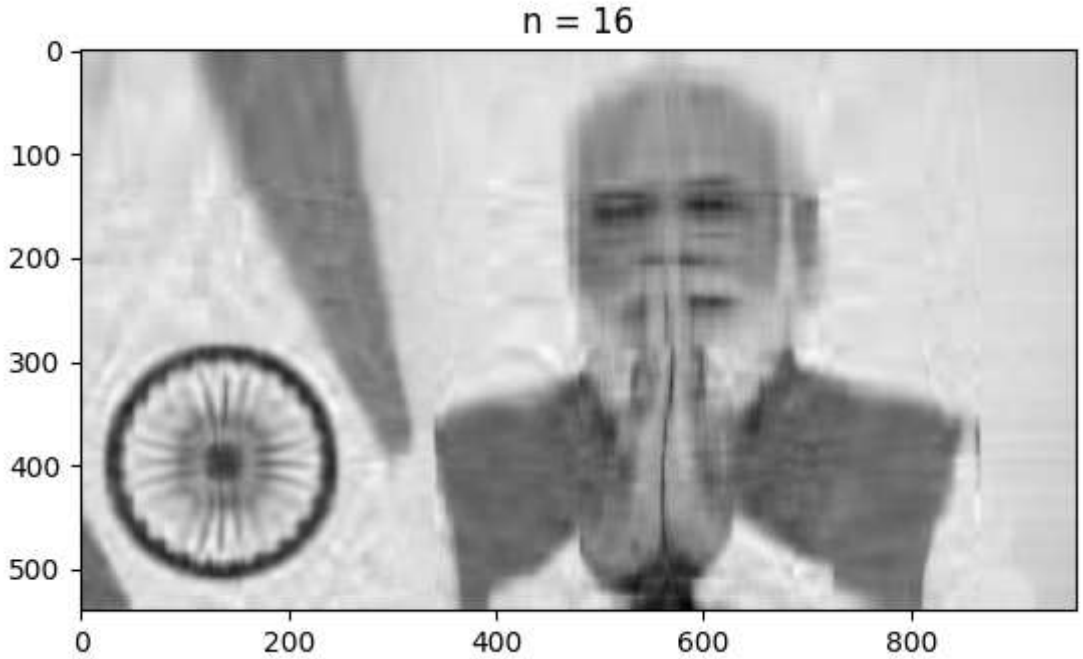


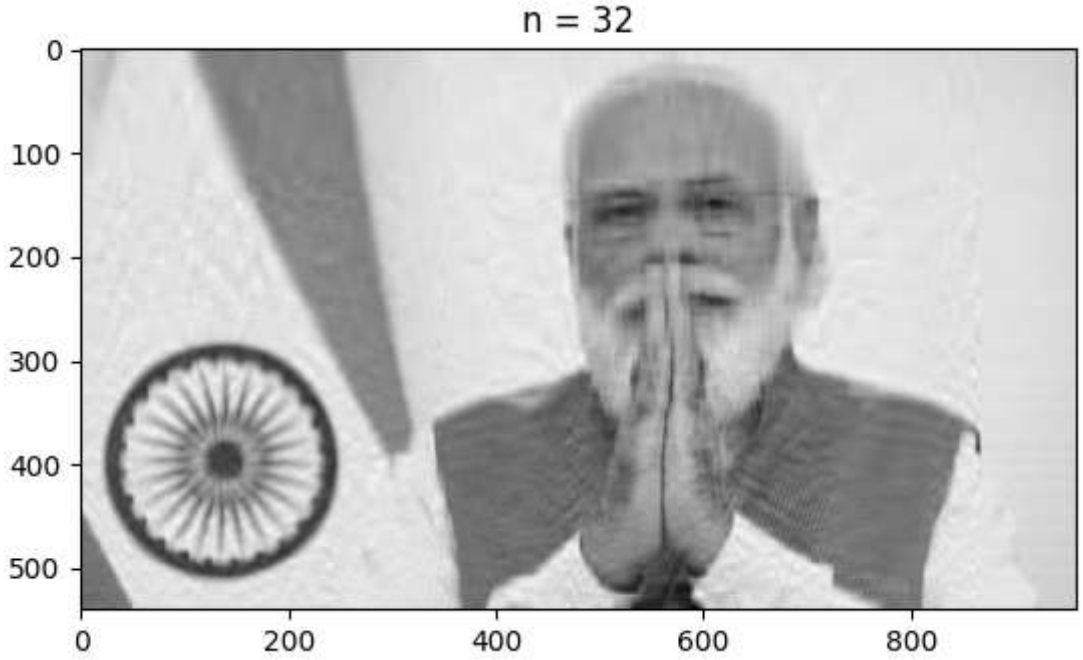
In [75]:
```
#more coloumns>> for more clear image>> construct a for loop
for i in [2,4,8,16,32,64]:
    reconstimg = np.matrix(U[:, :i]) *np.diag(sigma[:i]) * np.matrix(V[:i, :])
    plt.imshow(reconstimg, cmap='gray')
    title = "n = %s" % i
    plt.title(title)
    plt.show()
```

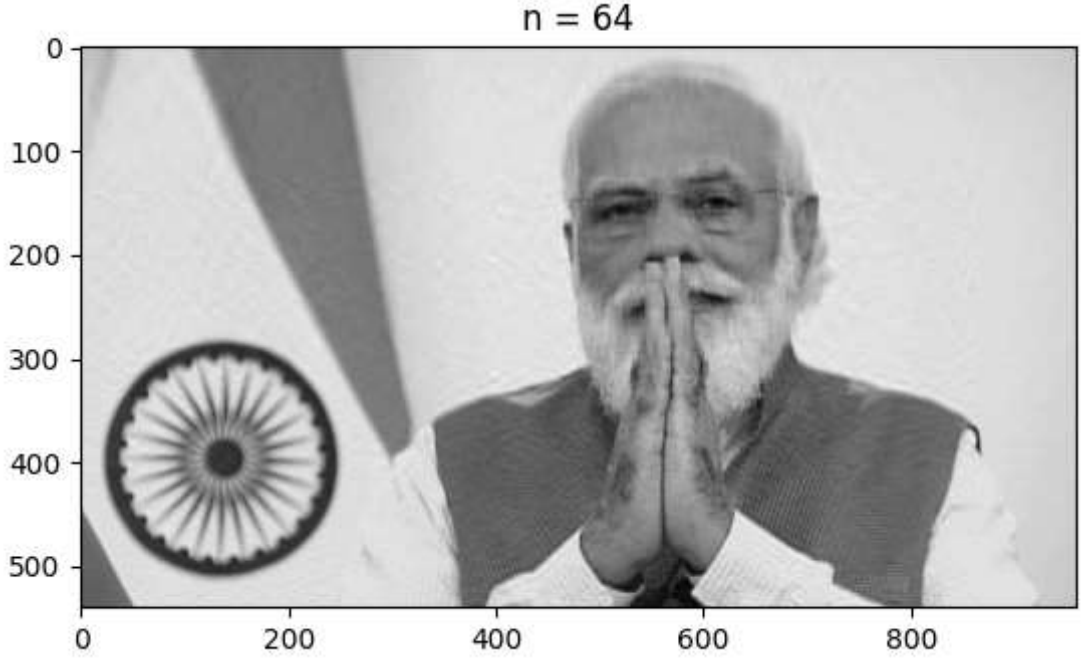

n = 2
0
100
200
300
400
500
0
200
400
600
800


n = 4
0
100
200
300
400
500
0
200
400
600
800


n = 8
0
100
200
300
400
500
0
200
400
600
800


n = 16
0
100
200
300
400
500
0
200
400
600
800


n = 32
0
100
200
300
400
500
0
200
400
600
800


n = 64
0
100
200
300
400
500
0
200
400
600
800

In [76]:
```
#dimensionality redn pca =1 means it generates 1 coloumn
#pca is feature extraction technique
```