In [ ]:
```
import sys
IN_COLAB = 'google.colab' in sys.modules
if IN_COLAB:
    !git clone https://github.com/cs357/demos-cs357.git
    !mv demos-cs357/figures/ .
```

Computing the SVD

In [1]:
```
import numpy as np
import numpy.linalg as la


import matplotlib.pyplot as plt
%matplotlib inline

from PIL import Image
```

1) For a square matrix

In [2]:
```
m = 4
n = m
A = np.random.randn(m, n)
print(A)
```

```
[[ 0.44751709  0.71846408  0.05114268  0.93548259]
 [ 0.3143188  -0.14512926  1.01478714  0.29193142]
 [-0.54443428 -1.31670691 -1.64647715 -0.61795394]
 [ 2.09187857 -1.21962118 -1.54855741 -0.73072719]]
```

Using numpy.linalg.svd

In [3]:
```
U, S, Vt = la.svd(A)
```

In [4]:
```
print(U)
print(U.shape)
```

```
[[ 0.16508158 -0.38626634 -0.77171984  0.47748808]
 [ 0.17725022 -0.28992475  0.62038655  0.70685679]
 [-0.53979397  0.65485407 -0.12527357  0.51390189]
 [-0.80619656 -0.58129864  0.06225385 -0.09090356]]
(4, 4)
```

In [5]:
```
print(Vt)
print(Vt.T.shape)
```

```
[[-0.37291595  0.52760245  0.68663979  0.33330939]
 [-0.92436364 -0.19565619 -0.24763231 -0.21435659]
 [ 0.04669702 -0.53957828  0.67992767 -0.49434101]
 [-0.06561201 -0.62626638  0.07001722  0.77368151]]
(4, 4)
```

In [6]:
```
print(S)
print(S.shape)
```

```
[3.38680354 1.98679465 1.02944611 0.519456  ]
(4,)
```

Using eigen-decomposition

Now compute the eigenvalues and eigenvectors of $A^T A$ as `eigvals` and `eigvecs`

In [7]:
```
eigvals, eigvecs = la.eig(A.T.dot(A))
```

Eigenvalues are real and positive. Coincidence?

In [8]:
```
eigvals
```

Out[8]:
```
array([11.4704382 ,  3.94735298,  0.26983454,  1.05975929])
```

`eigvecs` are orthonormal! Check:

In [9]:
```
eigvecs.T @ eigvecs
```

Out[9]:
```
array([[ 1.00000000e+00, -4.02455846e-16,  2.22044605e-16,
        -5.55111512e-17],
       [-4.02455846e-16,  1.00000000e+00, -1.94289029e-16,
        -3.74700271e-16],
       [ 2.22044605e-16, -1.94289029e-16,  1.00000000e+00,
        -3.88578059e-16],
       [-5.55111512e-17, -3.74700271e-16, -3.88578059e-16,
         1.00000000e+00]])
```

Now piece together the SVD:

In [10]:
```
S2 = np.sqrt(eigvals)
V2 = eigvecs
U2 = A @ V2 @ la.inv(np.diag(S2))
```

2) For a non-square square matrix

In [11]:
```
m = 3
n = 5
A = np.random.randn(m, n)
print(A)
```

```
[[-0.32401999  0.330958   -0.26428873 -1.00134046  0.12166824]
 [ 0.01996496 -0.56026826  0.82653363 -0.24298366  1.47013803]
 [-0.26083292 -0.51391272  0.03832262  0.60425759  0.42591705]]
```

You can obtain the SVD in the full format using `full_matrices=True` (full_matrices=True is the default value)

In [12]:
```
U, S, Vt = la.svd(A,full_matrices=True)
```

In [13]:
```
print(U)
print(U.shape)

print(Vt)
print(Vt.T.shape)

print(S)
print(S.shape)
```

```
[[-0.08763067 -0.85156735  0.51686934]
 [ 0.94633759 -0.23319346 -0.2237543 ]
 [ 0.3110724   0.46952514  0.82630509]]
(3, 3)
[[-0.01815185 -0.3855936   0.43824793  0.02454447  0.81137193]
 [ 0.11539148 -0.30435516  0.03901468  0.92520466 -0.19112023]
 [-0.76094432 -0.25181638 -0.56928227  0.07091078  0.16864642]
 [-0.6012135   0.5075857   0.57722198  0.19902169 -0.09002352]
 [ 0.21415273  0.66153668 -0.38619825  0.31425457  0.51826904]]
(5, 5)
[1.86483656 1.28953649 0.50919733]
(3,)
```

Check the eigen decomposition:

In [14]:
```
eigvals, eigvecs = la.eig(A.T.dot(A))
print(eigvals)
```

```
[3.47761540e+00 1.66290437e+00 2.59281920e-01 7.09446201e-17
 2.26776847e-16]
```

In [15]:
```
eigvals.sort()
```

In [16]:
```
np.sqrt(eigvals[-3:])
```

Out[16]:
```
array([0.50919733, 1.28953649, 1.86483656])
```

Or you can use get the reduced form of the SVD:

In [17]:
```
U, S, Vt = la.svd(A,full_matrices=False)
```

In [18]:
```
print('A = ', A.shape)
print('U = ', U.shape)
print('S = ', S.shape)
print('V = ', Vt.T.shape)
```

```
A =  (3, 5)
U =  (3, 3)
S =  (3,)
V =  (5, 3)
```

Relative cost of matrix factorizations

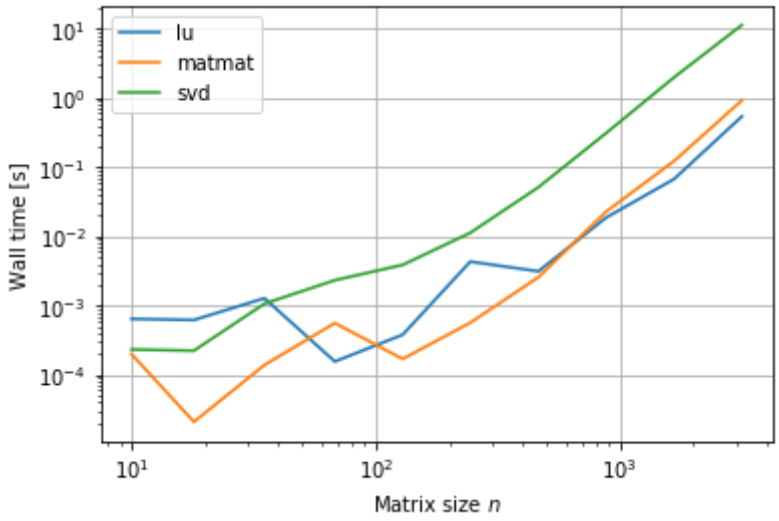
In [19]:
```
import numpy.linalg as npla
import scipy.linalg as spla
from time import time
```

In [20]:
```
n_values = np.logspace(1,3.5,10).astype(np.int32)
n_values
```

Out[20]:
```
array([  10,   18,   35,   68,  129,  244,  464,  879, 1668, 3162],
      dtype=int32)
```

In [21]:
```
def matmat(A):
    A @ A

for name, f in [
        ("lu", spla.lu_factor),
        ("matmat", matmat),
        ("svd", npla.svd)
        ]:

    times = []
    print("----->", name)

    for n in n_values:
        A = np.random.randn(n, n)

        start_time = time()
        f(A)
        delta_time = time() - start_time
        times.append(delta_time)

        print("%d - %f" % (n, delta_time))

    plt.loglog(n_values, times, label=name)

plt.legend(loc="best")
plt.xlabel("Matrix size $n$")
plt.ylabel("Wall time [s]");
plt.grid();
```

```
-----> lu
10 - 0.000645
18 - 0.000622
35 - 0.001281
68 - 0.000156
129 - 0.000381
244 - 0.004333
464 - 0.003141
879 - 0.018734
1668 - 0.067521
3162 - 0.542564
-----> matmat
10 - 0.000199
18 - 0.000021
35 - 0.000139
68 - 0.000561
129 - 0.000170
244 - 0.000568
464 - 0.002579
879 - 0.022818
1668 - 0.122959
3162 - 0.925557
-----> svd
10 - 0.000233
18 - 0.000224
35 - 0.001069
68 - 0.002334
129 - 0.003886
244 - 0.011294
464 - 0.051450
879 - 0.311695
1668 - 1.992078
3162 - 11.369725
```



SVD Applications

1) Rank of a matrix

Creating matrices for the examples:

In [22]:
```
m = 6
n = 4
# Creating the orthogonal U and Vt matrices
X = np.random.randn(m, m)
U, _ = la.qr(X)
X = np.random.randn(n, n)
Vt, _ = la.qr(X)


# Creating the singular values
S = np.zeros((m,n))
# This creates a full rank matrix
r = min(m,n)
# This creates a rank deficient matrix
r = np.random.randint(1,min(m,n))

print("the rank of A is = ",r)
# Completing the singular value matrix Sigma
sig = np.random.randint(1,50,r)
sig.sort()
sigmas = sig[::-1]
for i,s in enumerate(sigmas):
    S[i,i] = s

print(S)

print(U)

print(Vt)

A = U@S@Vt
```

```
the rank of A is =  3
[[18.  0.  0.  0.]
 [ 0. 16.  0.  0.]
 [ 0.  0.  9.  0.]
 [ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]]
[[-0.24349976 -0.86498521 -0.02758344  0.15659001  0.40779269 -0.030533
42]
 [ 0.41139505 -0.21292437  0.19845202  0.01524539 -0.13455003  0.853052
16]
 [-0.47628149 -0.15489593 -0.55914855 -0.36188646 -0.4931067   0.249800
04]
 [-0.44370751  0.40408518 -0.05522994 -0.10535011  0.66138176  0.433893
98]
 [-0.541059   -0.02656703  0.78098519 -0.04899051 -0.30589914  0.025241
46]
 [-0.2345138   0.13594263 -0.18498818  0.91147497 -0.20358556  0.141663
51]]
[[-0.6234891   0.23695507 -0.58955916 -0.45555859]
 [ 0.50205656 -0.5762697  -0.61656504 -0.18894443]
 [ 0.59028646  0.66928332 -0.0066165  -0.45119615]
 [ 0.10374219  0.40475347 -0.52175068  0.74376637]]
```

In [23]:
```
la.svd(A)
```

Out[23]:
```
(array([[ 0.24349976, -0.86498521, -0.02758344, -0.25782678, -0.3514416
9,
          0.04197245],
        [-0.41139505, -0.21292437,  0.19845202,  0.76734826, -0.3099743
2,
          0.24723811],
        [ 0.47628149, -0.15489593, -0.55914855,  0.54515779,  0.3356865
9,
         -0.16319801],
        [ 0.44370751,  0.40408518, -0.05522994,  0.10135996, -0.7758646
2,
         -0.15668031],
        [ 0.541059  , -0.02656703,  0.78098519,  0.17327305,  0.2558572
8,
          0.03354074],
        [ 0.2345138 ,  0.13594263, -0.18498818, -0.08489753,  0.0170387
9,
          0.94064039]]),
 array([1.80000000e+01, 1.60000000e+01, 9.00000000e+00, 8.78301847e-1
6]),
 array([[ 0.6234891 , -0.23695507,  0.58955916,  0.45555859],
        [ 0.50205656, -0.5762697 , -0.61656504, -0.18894443],
        [ 0.59028646,  0.66928332, -0.0066165 , -0.45119615],
        [-0.10374219, -0.40475347,  0.52175068, -0.74376637]]))
```

In [24]:
```
la.matrix_rank(A)
```

Out[24]:
```
3
```

2) Low-rank approximations

In [68]:
```
with Image.open("figures/quad.jpg") as img:
    rgb_img = np.array(img)
rgb_img.shape
```

Out[68]:
```
(500, 1417, 3)
```

In [69]:
```
img = np.sum(rgb_img, axis=-1)
img.shape
```

Out[69]:
```
(500, 1417)
```

In [70]:
```
plt.figure(figsize=(20,10))
plt.imshow(img, cmap="gray")
```

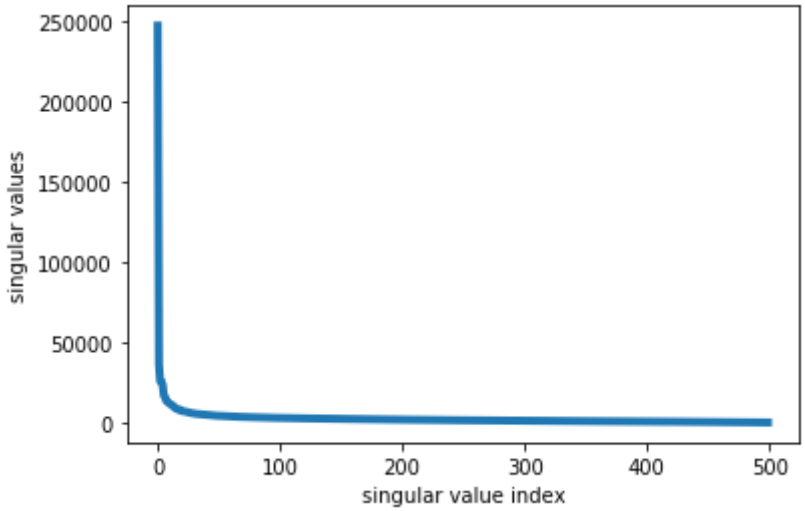
Out[70]:
```
<matplotlib.figure.AxesImage at 0x616f74cd0>
```

In [71]:
```
u, sigma, vt = np.linalg.svd(img)
print('A = ', img.shape)
print('U = ', u.shape)
print('S = ', sigma.shape)
print('V.T = ', vt.shape)
```

```
A =  (500, 1417)
U =  (500, 500)
S =  (500,)
V.T =  (1417, 1417)
```

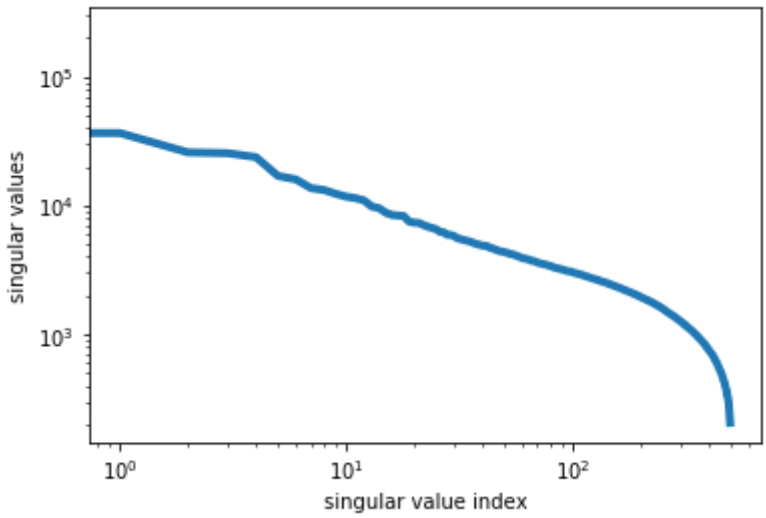
In [72]:
```
plt.plot(sigma, lw=4)
plt.xlabel('singular value index')
plt.ylabel('singular values')
```

Out[72]:
```
Text(0, 0.5, 'singular values')
```



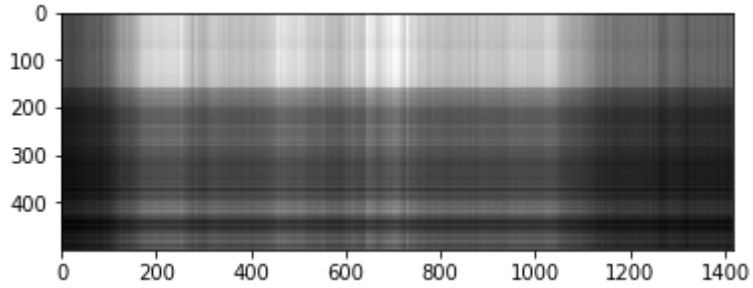
In [73]:
```
plt.loglog(sigma, lw=4)
plt.xlabel('singular value index')
plt.ylabel('singular values')
```

Out[73]:
```
Text(0, 0.5, 'singular values')
```



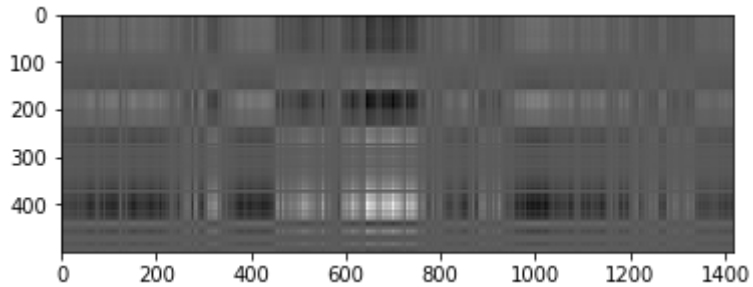
In [75]:
```
k = 0
img_k = sigma[k]*np.outer(u[:,k],vt[k,:])
plt.imshow(img_k, cmap="gray")
```

Out[75]:
```
<matplotlib.image.AxesImage at 0x6175a6b50>
```



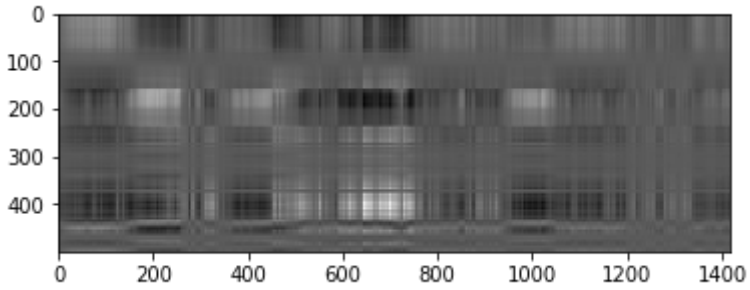
In [77]:
```
k = 1
img_k = sigma[k]*np.outer(u[:,k],vt[k,:])
plt.imshow(img_k, cmap="gray")
```

Out[77]:
```
<matplotlib.image.AxesImage at 0x618b12a90>
```



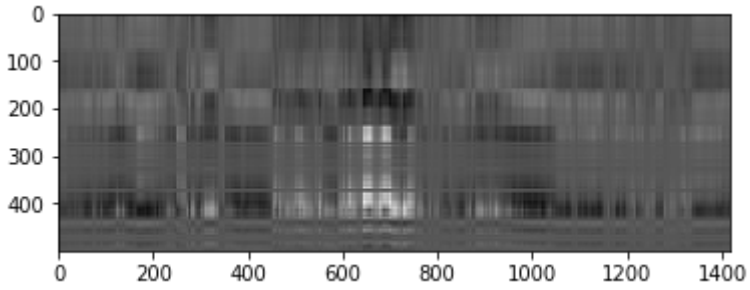
In [78]:
```
img_k = sigma[1]*np.outer(u[:,1],vt[1,:]) + sigma[2]*np.outer(u[:,2],vt[
2,:])
plt.imshow(img_k, cmap="gray")
```

Out[78]: <matplotlib.image.AxesImage at 0x61926d650>



In [79]:
```
img_k = sigma[1]*np.outer(u[:,1],vt[1,:]) + sigma[4]*np.outer(u[:,4],vt[
4,:])
plt.imshow(img_k, cmap="gray")
```

Out[79]: <matplotlib.image.AxesImage at 0x61933dd50>

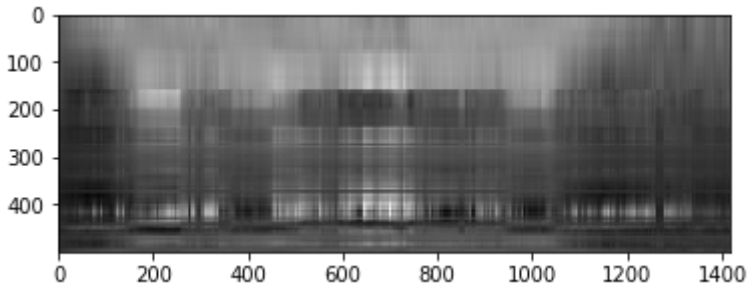


In [80]:
```
la.matrix_rank(img_k)
```

Out[80]: 2

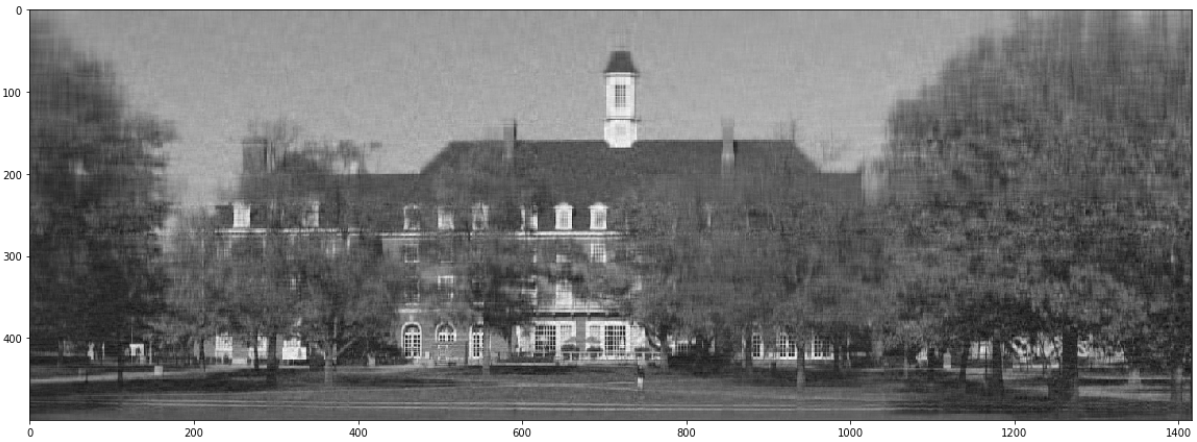
In [85]:
```
img_k = np.zeros(img.shape)
k=4
for i in range(k):
    img_k += sigma[i]*np.outer(u[:,i],vt[i,:])
plt.imshow(img_k, cmap="gray")
```

Out[85]: <matplotlib.image.AxesImage at 0x61ac49750>



In [86]:
```
k=50
compressed_img = u[:,:k+1] @ np.diag(sigma[:k+1]) @ vt[:k+1,:]
plt.figure(figsize=(20,10))
plt.imshow(compressed_img, cmap="gray")
```

Out[86]:
```
<matplotlib.image.AxesImage at 0x617155f50>
```



In [87]:
```
error = la.norm(img - compressed_img,2)
print(error)
```

```
4310.027781650038
```

In [88]:
```
sigma[k:k+3]
```

Out[88]:
```
array([4372.09358531, 4310.02778165, 4288.77646113])
```

In [89]:
```
original_size = img.size
compressed_size = u[:,:k].size + sigma[:k].size + vt[:k,:].size
print("original size: %d" % original_size)
print("compressed size: %d" % compressed_size)
print("ratio: %f" % (compressed_size / original_size))
```

```
original size: 708500
compressed size: 95900
ratio: 0.135356
```

In [90]:
```
compressed_img = np.zeros(img.shape)
for k in range(500):
    compressed_img += sigma[k]*np.outer(u[:,k], vt[k,:])
    error = la.norm(img - compressed_img,2)
    if error < 4000:
        break
k
```

Out[90]:
```
58
```

3) Pseudo-inverse

Square matrices:

In [36]:
```
m = 4
n = 4
# Creating the orthogonal U and Vt matrices
X = np.random.randn(m, m)
U, _ = la.qr(X)
X = np.random.randn(n, n)
Vt, _ = la.qr(X)


# Creating the singular values
S = np.zeros((m,n))
# This creates a full rank matrix
r = min(m,n)
# This creates a rank deficient matrix
r = np.random.randint(1,min(m,n))

print("the rank of A is = ",r)
# Completing the singular value matrix Sigma
sig = np.random.randint(1,50,r)
sig.sort()
sigmas = sig[::-1]
for i,s in enumerate(sigmas):
    S[i,i] = s

A = U@S@Vt
```

```
the rank of A is =  2
```

In [37]:
```
la.inv(A)
```

Out[37]:
```
array([[ 1.61396564e+15, -2.04168995e+14, -3.85474727e+13,
         1.49297515e+14],
       [ 3.71014240e+15, -3.31364507e+14,  8.20195651e+14,
        -1.13287453e+14],
       [ 7.99188296e+15, -9.61370797e+14,  1.35922103e+14,
         5.75128727e+14],
       [ 5.68980258e+14,  7.86677176e+13,  9.78677425e+14,
        -4.45777096e+14]])
```

In [38]:
```
la.cond(A)
```

Out[38]:
```
5.625671353846833e+16
```

In [39]:
```
la.svd(A)
```

Out[39]:
```
(array([[ 0.04051588, -0.12981411, -0.9343878 ,  0.32928134],
        [ 0.74485967, -0.64338206,  0.06355857, -0.16493605],
        [ 0.20516962,  0.4143654 , -0.33760532, -0.81989597],
        [ 0.63359921,  0.63048326,  0.09435252,  0.43833834]]),
 array([4.5000000e+01, 3.5000000e+01, 1.2989988e-15, 7.9990453e-16]),
 array([[ 0.90230364,  0.27238628, -0.29789664, -0.15136528],
        [ 0.37562097, -0.69908169,  0.20797578,  0.57178646],
        [ 0.16960004, -0.32861844,  0.58144767, -0.72468225],
        [-0.12646288, -0.57367275, -0.7279576 , -0.35353139]]))
```

In [40]:
```
la.pinv(A)
```

Out[40]:
```
array([[-0.00058078,  0.00803053,  0.00856088,  0.01947078],
       [ 0.00283812,  0.01735942, -0.00703454, -0.00875793],
       [-0.00103959, -0.008754  ,  0.00110402, -0.00044794],
       [-0.00225702, -0.01301624,  0.00607926,  0.00816883]])
```

In [41]:
```
u,s,vt = la.svd(A)
```

In [42]:
```
s
```

Out[42]:
```
array([4.5000000e+01, 3.5000000e+01, 1.2989988e-15, 7.9990453e-16])
```

In [43]:
```
sinv = np.zeros(s.shape)
for i,si in enumerate(s):
    if si > 1e-10:
        sinv[i] = 1/si
sinv
```

Out[43]:
```
array([0.02222222, 0.02857143, 0.        , 0.        ])
```

In [44]:
```
vt.T@np.diag(sinv)@u.T
```

Out[44]:
```
array([[-0.00058078,  0.00803053,  0.00856088,  0.01947078],
       [ 0.00283812,  0.01735942, -0.00703454, -0.00875793],
       [-0.00103959, -0.008754  ,  0.00110402, -0.00044794],
       [-0.00225702, -0.01301624,  0.00607926,  0.00816883]])
```

Rectangular matrices

In [45]:
```
m = 6
n = 4
# Creating the orthogonal U and Vt matrices
X = np.random.randn(m, m)
U, _ = la.qr(X)
X = np.random.randn(n, n)
Vt, _ = la.qr(X)


# Creating the singular values
S = np.zeros((m,n))
# This creates a rank deficient matrix
r = np.random.randint(1,min(m,n))

print("the rank of A is = ",r)
# Completing the singular value matrix Sigma
sig = np.random.randint(1,50,r)
sig.sort()
sigmas = sig[::-1]
for i,s in enumerate(sigmas):
    S[i,i] = s

A = U@S@Vt
```

```
the rank of A is =  2
```

In [46]:
```
la.pinv(A)
```

Out[46]:
```
array([[ 0.01508571,  0.01133314, -0.01648325, -0.00271905, -0.0011290
2,
         0.00398396],
       [-0.00359352,  0.01770222, -0.00602479,  0.00300241,  0.0053401
3,
        -0.00656651],
       [ 0.0153065 , -0.00178204, -0.01024653, -0.0042917 , -0.0044467
6,
         0.00769912],
       [-0.01570243, -0.02349052,  0.02286101,  0.00148051, -0.0017315
7,
        -0.00092695]])
```

In [47]:
```
u,s,vt = la.svd(A,full_matrices=False)
sinv = np.zeros(s.shape)
for i,si in enumerate(s):
    if si > 1e-10:
        sinv[i] = 1/si
vt.T@np.diag(sinv)@u.T
```

Out[47]:
```
array([[ 0.01508571,  0.01133314, -0.01648325, -0.00271905, -0.0011290
2,
         0.00398396],
       [-0.00359352,  0.01770222, -0.00602479,  0.00300241,  0.0053401
3,
        -0.00656651],
       [ 0.0153065 , -0.00178204, -0.01024653, -0.0042917 , -0.0044467
6,
         0.00769912],
       [-0.01570243, -0.02349052,  0.02286101,  0.00148051, -0.0017315
7,
        -0.00092695]])
```

4) Matrix Norms and Condition number

Square and non-singular matrices

In [48]:
```
m = 4
n = 4
# Creating the orthogonal U and Vt matrices
X = np.random.randn(m, m)
U, _ = la.qr(X)
X = np.random.randn(n, n)
Vt, _ = la.qr(X)


# Creating the singular values
S = np.zeros((m,n))
# This creates a full rank matrix
r = min(m,n)
# This creates a rank deficient matrix
## r = np.random.randint(1,min(m,n))

print("the rank of A is = ",r)
# Completing the singular value matrix Sigma
sig = np.random.randint(1,50,r)
sig.sort()
sigmas = sig[::-1]
for i,s in enumerate(sigmas):
    S[i,i] = s

A = U@S@Vt
```

```
the rank of A is =  4
```

Given the SVD of A...

In [49]:
```
u,s,vt = la.svd(A)
s
```

Out[49]:
```
array([49., 47., 37.,  8.])
```

... determine the eucledian norm of A:

In [50]:
```
la.norm(A,2)
```

Out[50]:
```
49.00000000000002
```

... determine the eucledian norm of A^{-1}

In [51]:
```
1/9
```

Out[51]:
```
0.1111111111111111
```

In [52]:
```
la.norm(la.inv(A),2)
```

Out[52]:
```
0.12499999999999993
```

... determine the condition number of A:

In [53]:
```
33/9
```

Out[53]:
```
3.6666666666666665
```

In [54]:
```
la.cond(A)
```

Out[54]:
```
6.125000000000008
```

Square and singular matrices

In [55]:
```
m = 4
n = 4
# Creating the orthogonal U and Vt matrices
X = np.random.randn(m, m)
U, _ = la.qr(X)
X = np.random.randn(n, n)
Vt, _ = la.qr(X)


# Creating the singular values
S = np.zeros((m,n))
# This creates a full rank matrix
r = min(m,n)
# This creates a rank deficient matrix
r = np.random.randint(1,min(m,n))

print("the rank of A is = ",r)
# Completing the singular value matrix Sigma
sig = np.random.randint(1,50,r)
sig.sort()
sigmas = sig[::-1]
for i,s in enumerate(sigmas):
    S[i,i] = s

A = U@S@Vt
```

```
the rank of A is =  1
```

Given the SVD of A...

In [56]:
```
u,s,vt = la.svd(A)
s
```

Out[56]:
```
array([4.00000000e+00, 8.16175024e-16, 9.85243825e-17, 7.85887303e-18])
```

In [57]:
```
la.matrix_rank(A)
```

Out[57]:
```
1
```

... determine the eucledian norm of A:

In [58]:
```
33
```

Out[58]:
```
33
```

In [59]:
```
la.norm(A,2)
```

Out[59]:
```
4.000000000000003
```

... determine the eucledian norm of A^+

In [60]:
```
1/13
```

Out[60]:
```
0.07692307692307693
```

In [61]:
```
la.norm(la.pinv(A),2)
```

Out[61]:
```
0.2499999999999998
```

... determine the condition number of A:

In [62]:
```
np.inf
```

Out[62]:
```
inf
```

In [63]:
```
la.cond(A)
```

Out[63]:
```
5.089788297766977e+17
```

In [ ]:
```

```