Assignment 2 submission

Name: Duong Doan Tung

Student number: 21010294

Programming language used: Python

Programming environment used: Jupyter Notebook

Part 1

2.4 Compute the following products:

In [20]:
```
import numpy as np
#b
A = np.arange(1, 10).reshape(3, 3)
B = np.array([
    [1,1,0],
    [0,1,1],
    [1,0,1]
])
ans = np.dot(A, B)
print(ans)

#d
A = np.array([
    [1,2,1,2],
    [4,1,-4,-1],
])
B = np.array([
    [0,3],
    [1,-1],
    [2,1],
    [5,2]
])
ans = np.dot(A, B)
print(ans)
```

```
[[ 4  3  5]
 [10  9 11]
 [16 15 17]]
[[ 14   6]
 [-12   5]]
```

2.5b Find the set S of all solutions in x of the following inhomogeneous linear system Ax = b, where A and b are defined as follows:

In [21]:
```
import numpy.linalg as la
import scipy.linalg as spla
A = np.array([
    [1,-1,0,0,1],
    [1,1,0,-3,0],
    [2,-1,0,1,-1],
    [-1,2,0,-2,-1]
])
#remove the 3rd column
A = np.delete(A, 2, 1)
B = np.array([3,6,5,-1]).reshape(4,1)

X = np.dot(la.inv(A), B)
print(X)
```

```
[[4.]
 [0.]
 [4.]
 [0.]]
```

2.11 Write

```
y = [[1],
     [-2],
     [5]]
```

as a linear combination of the vectors

```
x1 = [[1],
      [1],
      [1]]
x2 = [[1],
      [2],
      [3]]
x3 = [[2],
      [-1],
      [1]]
```

In [22]:
```
x1 = np.array([[1],
      [1],
      [1]])
x2 = np.array([[1],
      [2],
      [3]])
x3 = np.array([[2],
      [-1],
      [1]])

#y is linear combination of x1, x2, x3
y = np.array([[1],[-2],[5]])

#solve for alpha1, alpha2, alpha3
X = np.hstack((x1, x2, x3))
A = np.dot(la.inv(X), y)
print(A.T)
```

```
[[-6.  3.  2.]]
```

Or in the mathematical form: -6x1 + 3x2 + 2x3 = y

3.3 Compute the distance between x,y

In [23]:
```
X = np.array([[1],
        [2],
        [3]])
Y = np.array([[-1],
        [-1],
        [0]])

#distance between X and Y using inner product (a)
dist = np.dot(X.T, Y)
print(dist[0,0])
#(b)
A = np.array([[2,1,0],
              [1,3,-1],
              [0,-1,2]])
dist = np.dot(np.dot(X.T, A), Y)
print(dist[0,0])
```

```
-3
-8
```

4.2 Compute the following determinants efficiently:

In [24]:
```
A = np.array([  [2,0,1,2,0],
                [2,-1,0,1,1],
                [0,1,2,1,2],
                [-2,0,2,-1,2],
                [2,0,0,1,1]
            ])
#deteminant of A
det = la.det(A)
print(det)
```

```
6.000000000000003
```

4.4 Comute all eigenspaces of the following matrix:

In [25]:
```
A = np.array([  [0,-1,1,1],
                [-1,1,-2,3],
                [2,-1,0,0],
                [1,-1,1,0]
            ])
#eigenspace of A
eig = la.eig(A)
print(eig)
```

```
(array([ 2.        ,  1.        , -1.00000002, -0.99999998]), array([[ 5.77350269e-01,  5.00000000e-01,  1.587
09825e-08,
         1.58709821e-08],
       [-1.44897623e-16,  5.00000000e-01, -7.07106773e-01,
         7.07106789e-01],
       [ 5.77350269e-01,  5.00000000e-01, -7.07106789e-01,
         7.07106773e-01],
       [ 5.77350269e-01,  5.00000000e-01,  3.20493788e-17,
        -6.19518495e-17]]))
```

4.7d Are the following matrices diagonalizable? If yes, determine their diagonal form and a basis with respect to which the transformation matrices are diagonal. If no, give reasons why they are not diagonalizable

In [26]:
```
'''
A =

5 -6 -6
-1 4 2
3 -6 -4
'''
A = np.array([  [5,-6,-6],
                [-1,4,2],
                [3,-6,-4]
            ])
#check if A diagonalizable or not
iA = la.inv(A)
eig = la.eig(A)

if np.allclose(np.dot(iA, A), np.diag(eig[0])):
    print('A is diagonalizable')
else:
    print('A is not diagonalizable')
```

```
A is not diagonalizable
```

4.9 Find the singular value decomposition

In [27]:
```
A = np.array([  [2,2],
                [-1,1]])
#find the singular value decomposition of A
U, s, V = la.svd(A)
print(U)
print(s)
print(V)
```

```
[[-1.00000000e+00  1.11022302e-16]
 [ 8.64164897e-17  1.00000000e+00]]
[2.82842712 1.41421356]
[[-0.70710678 -0.70710678]
 [-0.70710678  0.70710678]]
```

Part 2

A

In [28]:
```
import numpy as np
#Use arange to create a variable named foo that stores an array of numbers from 0 to 29, inclusive. Print foo
foo = np.arange(30)
print(foo)
print(foo.shape)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29]
(30,)
```

B

In [29]:
```
#Use the reshape function to change foo to a validly-shaped two-dimensional matrix and store it in a new varia
bar = foo.reshape(6,5)
print(bar)
print(bar.shape)
```

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]
 [20 21 22 23 24]
 [25 26 27 28 29]]
(6, 5)
```

C

In [30]:
```
baz = foo.reshape(2,3,5)
print(baz)
print(baz.shape)
```

```
[[[ 0  1  2  3  4]
  [ 5  6  7  8  9]
  [10 11 12 13 14]]

 [[15 16 17 18 19]
  [20 21 22 23 24]
  [25 26 27 28 29]]]
(2, 3, 5)
```

D

In [31]:
```
#D
print(bar)

#change first value row 2 to -1
bar[1,0] = -1
print(bar)

print(np.sum(bar, axis=1, keepdims=True))
```

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]
 [20 21 22 23 24]
 [25 26 27 28 29]]
[[ 0  1  2  3  4]
 [-1  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]
 [20 21 22 23 24]
 [25 26 27 28 29]]
[[ 10]
 [ 29]
 [ 60]
 [ 85]
 [110]
 [135]]
```

E

In [32]:
```
print(baz)
#Sum baz over its second dimension and print the result.
print(baz.sum(axis=1))
#Sum baz over its third dimension and print the result.
print(baz.sum(axis=2))

print(baz.sum(axis=0))
#Sum baz over both its first and third dimensions and print the result.
print(baz.sum(axis=(0,2)))
```

```
[[[ 0  1  2  3  4]
  [-1  6  7  8  9]
  [10 11 12 13 14]]

 [[15 16 17 18 19]
  [20 21 22 23 24]
  [25 26 27 28 29]]]
[[ 9 18 21 24 27]
 [60 63 66 69 72]]
[[ 10  29  60]
 [ 85 110 135]]
[[15 17 19 21 23]
 [19 27 29 31 33]
 [35 37 39 41 43]]
[ 95 139 195]
```

F

In [33]:
```
#F
print(bar)
#Slice out the second row of bar and print it.
print(bar[1,:])
#Slice out the last column of bar using the −1 notation and print it.
print(bar[:,-1])
#Slice out the top right 2 × 2 submatrix of bar and print it
print(bar[:2,3:])
```

```
[[ 0  1  2  3  4]
 [-1  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]
 [20 21 22 23 24]
 [25 26 27 28 29]]
[-1  6  7  8  9]
[ 4  9 14 19 24 29]
[[3 4]
 [8 9]]
```

Part 3

A

In [34]:
```
#create a vector 1, 2, . . . , 10 by adding 1 to the result of the arange function.
foo = np.arange(0,10)
foo = foo + 1
print(foo)
```

```
[ 1  2  3  4  5  6  7  8  9 10]
```

B

In [35]:
```
#, create a 10 × 10 matrix A in which Aij = i + j. You’ll be able to do this using the vector you just
#created, and adding it to a reshaped version of itself
print(foo.reshape(10,1) + foo)
```

```
[[ 2  3  4  5  6  7  8  9 10 11]
 [ 3  4  5  6  7  8  9 10 11 12]
 [ 4  5  6  7  8  9 10 11 12 13]
 [ 5  6  7  8  9 10 11 12 13 14]
 [ 6  7  8  9 10 11 12 13 14 15]
 [ 7  8  9 10 11 12 13 14 15 16]
 [ 8  9 10 11 12 13 14 15 16 17]
 [ 9 10 11 12 13 14 15 16 17 18]
 [10 11 12 13 14 15 16 17 18 19]
 [11 12 13 14 15 16 17 18 19 20]]
```

C

In [36]:
```
#A very common use of broadcasting is to standardize data, i.e., to make it have zero mean and unit variance.
#First, create a fake “data set” with 50 examples, each with five dimension

data = np.exp(np.random.randn(50,5))
```

D

In [37]:
```
#, compute the mean and standard deviation of each column. This should result in two vectors of length 5. You’
#vectors into variables and print both of them.

mean = np.mean(data, axis=0)
std = np.std(data, axis=0)
print(mean)
print(std)
```

```
[1.9849997  1.60560892 1.99496802 1.63900606 1.01935199]
[2.75443754 1.74432188 3.30689691 1.88321963 0.8300159 ]
```

E

In [38]:
```
#Now standardize the data matrix by 1) subtracting the mean off of each column, and 2) dividing each
#column by its standard deviation. Do this via broadcasting, and store the result in a matrix called
#normalized. To verify that you successfully did it, compute the mean and standard deviation of
#the columns of normalized and print them out

normalized = (data - mean)/std
print(np.mean(normalized, axis=0))
print(np.std(normalized, axis=0))
```

```
[-1.91513472e-17  1.66533454e-17  4.77395901e-17 -2.30926389e-16
  3.79696274e-16]
[1. 1. 1. 1. 1.]
```

Part 4

A

In [44]:
```
#Create a function that produce a vandermonde matrix
def vandermonde(N):
    vec = np.arange(1,N+1)
    return vec.reshape(N,1)**np.arange(N)
vander = vandermonde(12)
print(vander)
```

```
[[          1           1           1           1           1           1
            1           1           1           1           1           1]
 [          1           2           4           8          16          32
           64         128         256         512        1024        2048]
 [          1           3           9          27          81         243
          729        2187        6561       19683       59049      177147]
 [          1           4          16          64         256        1024
         4096       16384       65536      262144     1048576     4194304]
 [          1           5          25         125         625        3125
        15625       78125      390625     1953125     9765625    48828125]
 [          1           6          36         216        1296        7776
        46656      279936     1679616    10077696    60466176   362797056]
 [          1           7          49         343        2401       16807
       117649      823543     5764801    40353607   282475249  1977326743]
 [          1           8          64         512        4096       32768
       262144     2097152    16777216   134217728  1073741824           0]
 [          1           9          81         729        6561       59049
       531441     4782969    43046721   387420489  -808182895  1316288537]
 [          1          10         100        1000       10000      100000
      1000000    10000000   100000000  1000000000  1410065408  1215752192]
 [          1          11         121        1331       14641      161051
      1771561    19487171   214358881 -1937019605   167620825  1843829075]
 [          1          12         144        1728       20736      248832
      2985984    35831808   429981696   864813056  1787822080   -20971520]]
```

B

In [45]:
```
#Now, let’s make a pretend linear system problem with this matrix. Create a vector of all ones, of length 12 a
#that in a new vector and call it b. Print the vector b.

x = np.ones(12)
b = np.dot(vander, x)
print(b)
```

```
[1.20000000e+01 4.09500000e+03 2.65720000e+05 5.59240500e+06
 6.10351560e+07 4.35356467e+08 2.30688120e+09 1.22713351e+09
 9.43953692e+08 3.73692871e+09 3.10225064e+08 3.10073456e+09]
```

C

In [46]:
```
#First, solve the linear system the naïve way, pretending like you don’t know x. Import numpy.linalg,
#invert V and multiply it by b. Print out your result. What should you get for your answer? If the answer is
#different than what you expected, write a sentence about that difference.

x = np.dot(la.inv(vander), b)
print(x)
```

```
[0.997715   1.00320816 0.99892426 1.00016022 0.9999876  1.
 1.00000007 1.         1.         1.         1.         1.        ]
```

D

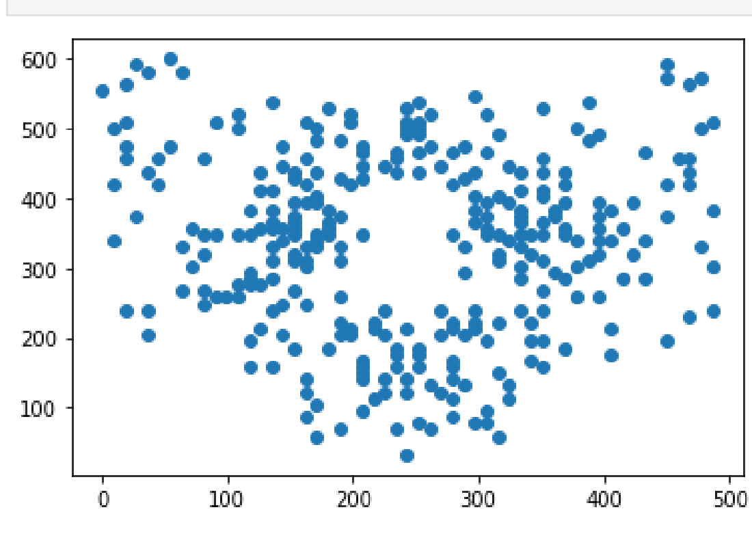
In [47]:
```
#Now, solve the same linear system using solve. Print out the result. Does it seem more or less in line
#with what you’d expect

x = la.solve(vander, b)
print(x)
```

```
[1.00002643 0.99993457 1.00006231 0.99996941 1.00000858 0.99999858
 1.00000014 0.99999999 1.         1.         1.         1.        ]
```

Part 5

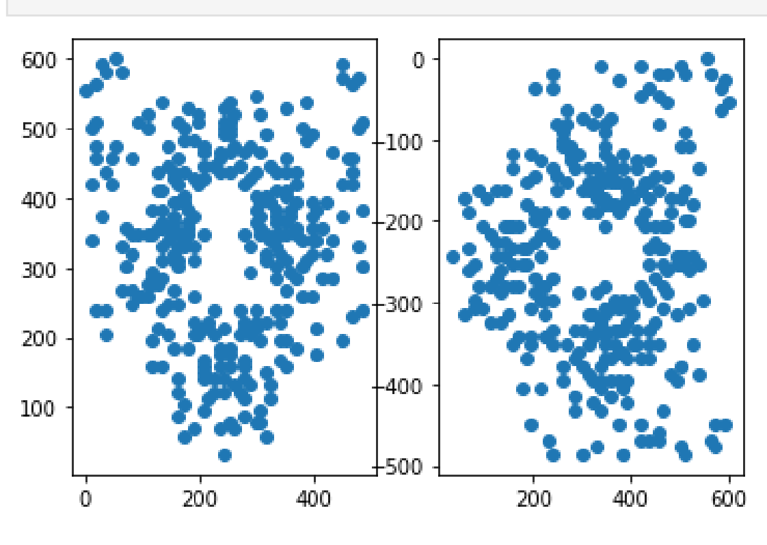
A

In [48]:
```
#load the coords.pkl and plot the points (file have a numpy array of 2D points 296x2)
import pickle
import matplotlib.pyplot as plt
with open('coords.pkl', 'rb') as f:
    coords = pickle.load(f)
plt.scatter(coords[:,0], coords[:,1])
plt.show()
```

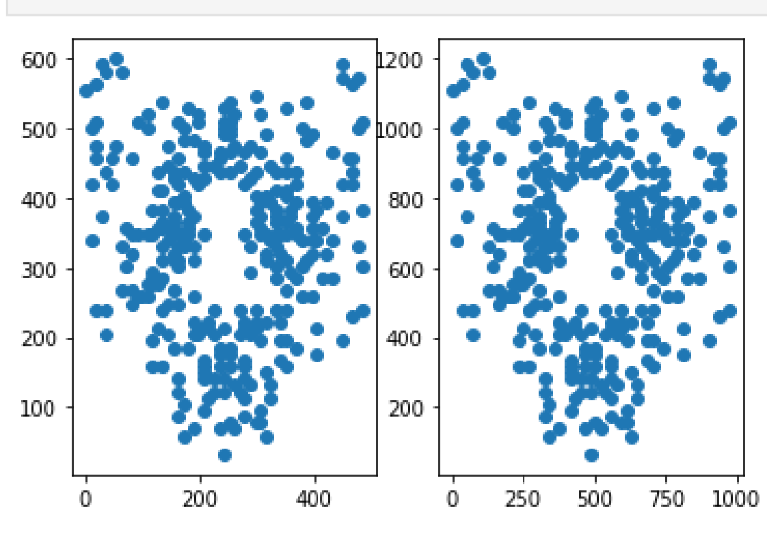


B

In [53]:
```
omega = np.pi/2
rotated = np.array([[np.cos(omega), -np.sin(omega)],
                    [np.sin(omega), np.cos(omega)]])
rotated_coords = np.dot(coords, rotated)
#create a sub plot with 3 plots
fig, ax = plt.subplots(1,2)
ax[0].scatter(coords[:,0], coords[:,1])
ax[1].scatter(rotated_coords[:,0], rotated_coords[:,1])
plt.show()
```



In [54]:
```
scale_factor = (2,2)
scaled = np.array([[scale_factor[0], 0],
                   [0, scale_factor[1]]])
scaled_coords = np.dot(coords, scaled)
fig, ax = plt.subplots(1,2)
ax[0].scatter(coords[:,0], coords[:,1])
ax[1].scatter(scaled_coords[:,0], scaled_coords[:,1])
plt.show()
```



In [55]:
```
shear_factor = (1,1)
shear = np.array([[1, shear_factor[0]],
                  [shear_factor[1], 1]])
sheared_coords = np.dot(coords, shear)
fig, ax = plt.subplots(1,2)
ax[0].scatter(coords[:,0], coords[:,1])
ax[1].scatter(sheared_coords[:,0], sheared_coords[:,1])

plt.show()
```