svd_transformation

February 21, 2019

1 Readme

1.1 Places where solutions are required are marked with #TODO

1.2 You will not need to modify any section not marked as #TODO to answer this question.

1.3 Make sure the helper file. svd_transformation_helper.py is in the same folder as this .ipynb

1.4 Make sure you have numpy, matplotlib and itertools packages installed for python

1.5 Q3b has 3 subparts i, ii, and iii

1.6 Q3c has 4 subparts i, ii, iii and iv

1.7 Q3d has 2 subparts i,ii

1.8 Q3e has only 1 subpart

In [7]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from svd_transformation_helper import visualize_function
from svd_transformation_helper import matrix_equals, is_orthogonal
```

In [8]:
```
DISABLE_CHECKS = False #Set this to True only if you get Value Errors about inputs eve
#when you are sure that what you are inputting is correct.
#WARNING: Setting this to True and entering wrong inputs can lead to all kinds of craz


def visualize(U = np.identity(2), D = np.ones(2), VT = np.identity(2), num_grid_points_
    disable_checks = DISABLE_CHECKS, show_original = True, show_VT = True, show_DVT = T
    '''
    Inputs:
    A has singular value decomposition A = U np.diag(D) VT
    U: 2 x 2 orthogonal matrix represented as a np.array of shape (2,2)
    D: Diagonal entries corresponding to the diagonal matrix in SVD represented as a np
    VT: 2 x 2 orthogonal matrix represented as a np.array of shape (2,2)
```

```
num_grid_points_per_dim: Spacing of points used to represent circle (Decrease this
disable_checks: If False then have checks in  place to make sure dimensions of VT,
show_original: If True plots original unit circle and basis vectors
show_VT: If True plots transformation by VT
show_DVT: If True plots transformation by DVT
show_UDVT: If True plots transformation by UDVT
'''

visualize_function(U=U, D=D, VT=VT, num_grid_points_per_dim=num_grid_points_per_dim
                  show_original=show_original, show_VT=show_VT, show_DVT=show_DVT, s
```

2 We start by looking at transformation by V^T, D, U separately.

3 Effect of the linear transformation by orthogonal matrix V^T

A 2 x 2 orthogonal matrix can be viewed as a linear transformation that performs some combination of rotations and reflections. Note that both rotation and reflection are operations that preserve length of vectors and angle between vectors.

3.1 V^T as a rotation matrix

First we set V^T as a counter-clockwise rotation matrix.

3.2 Q3b i) Fill in the function "get_RCC(theta)" to return a 2 x 2 matrix that when applied to a vector x rotates it by theta radians counter clockwise.

Example: If $V^T = RCC\left(\frac{\pi}{4}\right)$ and x = $\begin{bmatrix}1\\0\end{bmatrix}$, then,

$$V^T\begin{bmatrix}1\\0\end{bmatrix} = \begin{bmatrix}\frac{1}{\sqrt{2}}\\\frac{1}{\sqrt{2}}\end{bmatrix}$$

In [13]:
```
def get_RCC(theta):
    '''
    Returns a 2 x 2 orthogonal matrix that rotates x by theta radians counter-clockwi
    '''

    RCC = np.array([[np.cos(theta), -np.sin(theta)], [np.sin(theta), np.cos(theta)]])

    ##################################################################################
    #Some assertions (WARNING: Do not modify below code)
    if DISABLE_CHECKS is False:
        if not isinstance(RCC, np.ndarray):
                raise ValueError('RCC must be a np.ndarray')
        if len(RCC.shape) != 2 or (RCC.shape != np.array([2,2])).any():
                raise ValueError('RCC must have shape [2,2]')
    return RCC
```

3.3 #TODO Fill in solution to Q3b i. in cell above

3.3.1 get_RCC(theta) function test

If the function get_RCC(theta) is defined correctly then you should not get any ERROR statement here.

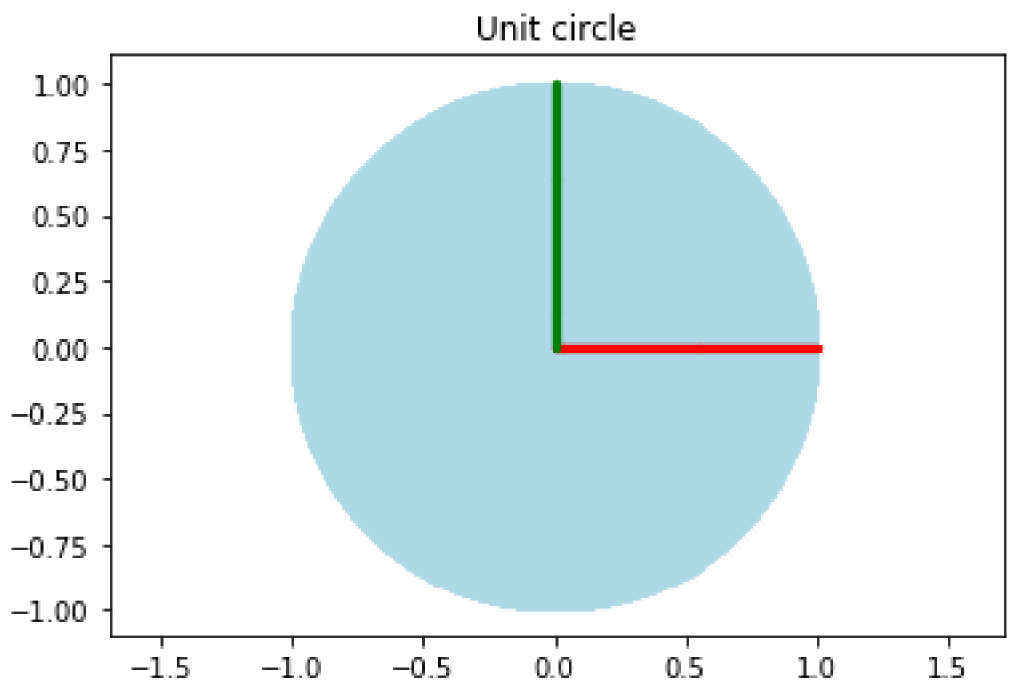
In [14]:
```
x = np.array([[1,0]]).T
V_test = get_RCC(np.pi/4)
y = np.matmul(V_test, x)
expected_y = np.array([[1/np.sqrt(2), 1/np.sqrt(2)]]).T
print("y:")
print(y)
print("Expected y:")
print(expected_y)
if not matrix_equals(y, expected_y):
    print("ERROR: y does not match expected_y. Check if function get_RCC(theta) is cor
else:
    print("MATCHED: y matches expected_y!")
```

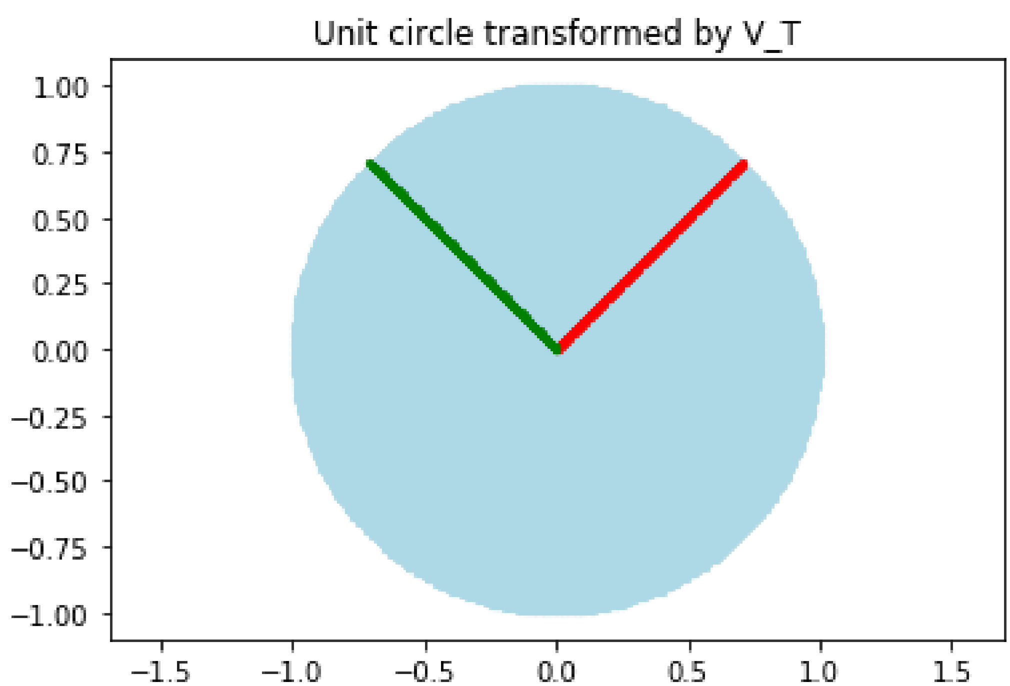
```
y:
[[0.70710678]
 [0.70710678]]
Expected y:
[[0.70710678]
 [0.70710678]]
MATCHED: y matches expected_y!
```

Next we observe how V^T transforms the unit circle and unit basis vectors when:

1) $V^T = RCC\left(\frac{\pi}{4}\right)$

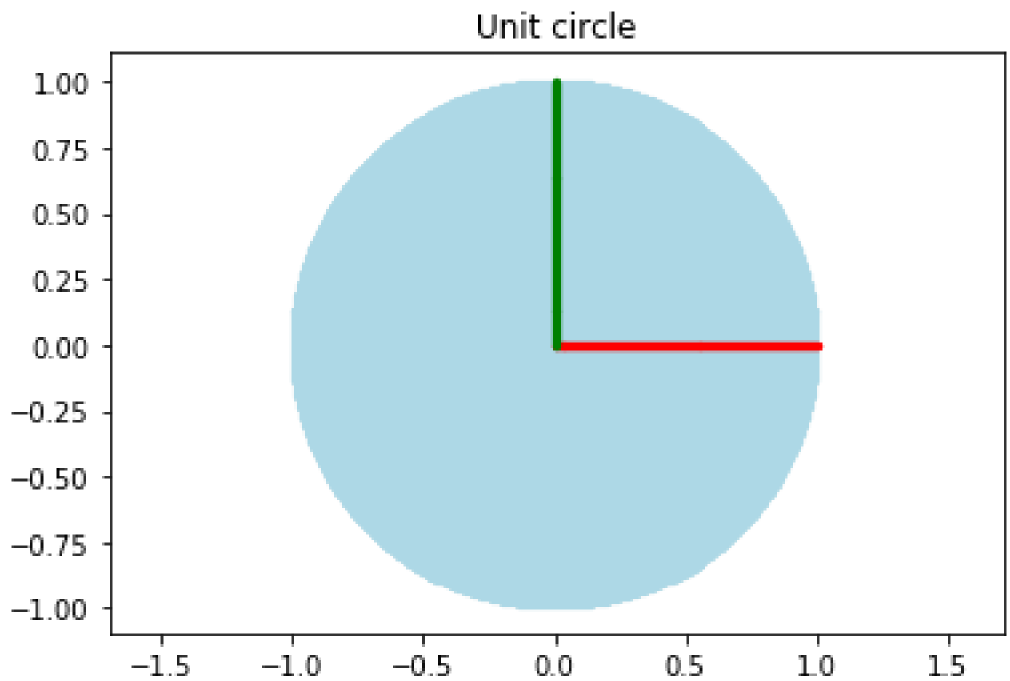
In [15]:
```
VT_1 = get_RCC(np.pi/4)
visualize(VT = VT_1, show_DVT=False, show_UDVT=False)
```

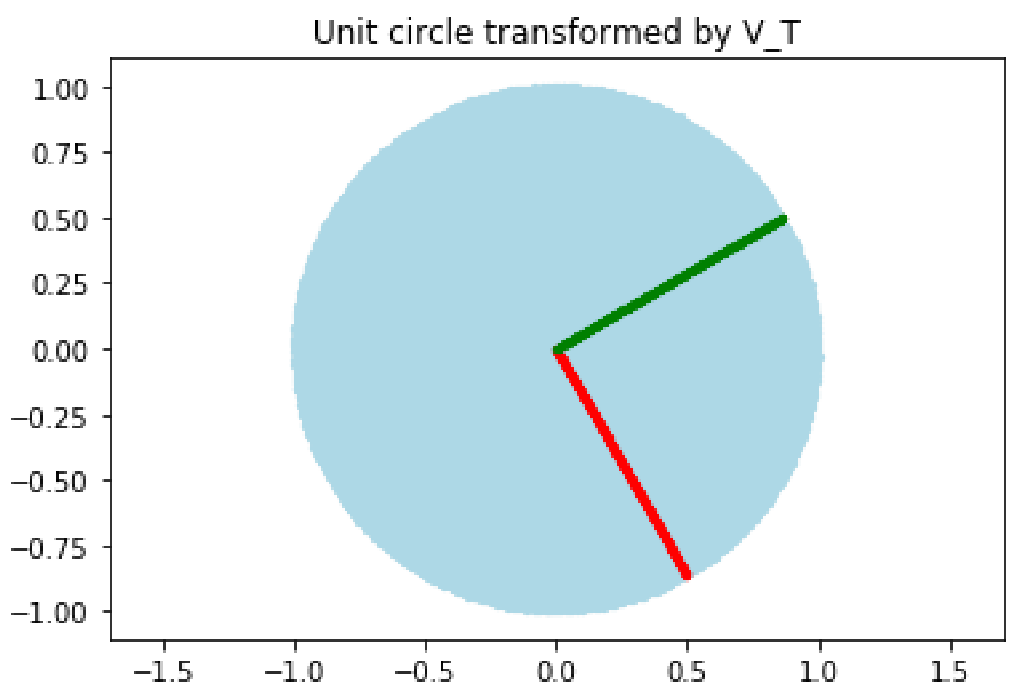




2) $V^T = RCC\left(\frac{-\pi}{3}\right)$

```
In [16]: VT_2 = get_RCC(-np.pi/3)
         visualize(VT = VT_2, show_DVT=False, show_UDVT=False)
```





Next we consider the case where V^T is a reflection matrix.

3.4 V^T as a relfection matrix

A reflection matrix is another type of orthogonal matrix.

3.5 Q3b ii) Fill in the function "get_RFx()" to return a 2 x 2 matrix that when applied to a vector x reflects it about the x-axis.

Example: If $V^T = RFx()$ and $x = \begin{bmatrix}1\\1\end{bmatrix}$, then,

$$V^T \begin{bmatrix}1\\1\end{bmatrix} = \begin{bmatrix}1\\-1\end{bmatrix}$$

```
In [19]: def get_RFx():
             '''
             Returns a 2 x 2 orthogonal matrix that reflects about x-axis
             '''


             RFx = np.array([[1,0],[0,-1]]) #TODO: Solution to Q3b ii) change this line by fil

             ################################################################################
             #Some assertions (WARNING: Do not modify below code)
             if DISABLE_CHECKS is False:
                 if not isinstance(RFx, np.ndarray):
                         raise ValueError('RFx must be a np.ndarray')
                 if len(RFx.shape) != 2 or (RFx.shape != np.array([2,2])).any():
                         raise ValueError('RFx must have shape [2,2]')
             return RFx
```

3.6 #TODO Fill in solution to Q3b ii. in cell above

3.6.1 get_RFx() function test

If the function get_RFx() is defined correctly then you should see a MATCHED statement here.

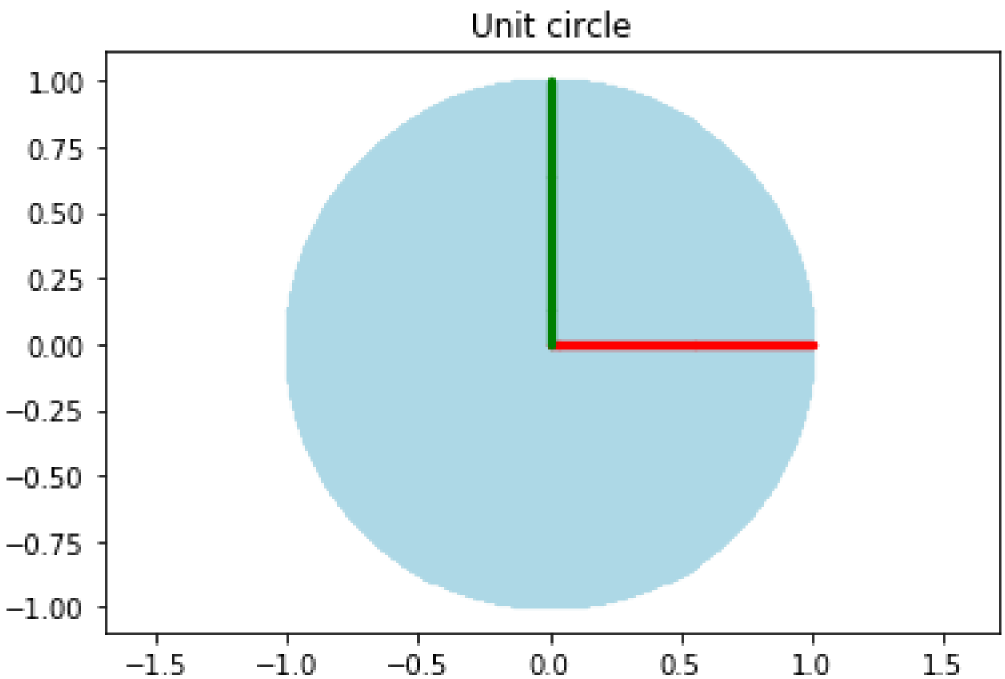
```
In [20]: x = np.array([[1,1]]).T
         V_test = get_RFx()
         y = np.matmul(V_test, x)
         expected_y = np.array([[1, -1]]).T
         print("y:")
         print(y)
         print("Expected y:")
         print(expected_y)
         if not matrix_equals(y, expected_y):
             print("ERROR: y does not match expected_y. Check if function get_RFx() is complete
         else:
             print("MATCHED: y matches expected_y!")
```

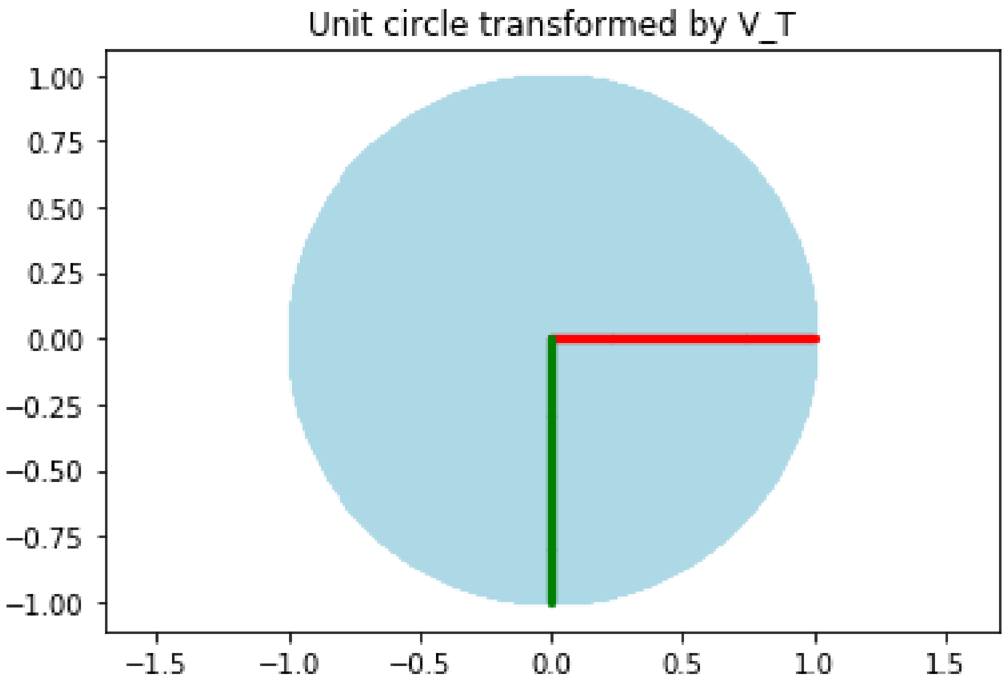
```
y:
[[ 1]
```

```
 [-1]]
Expected y:
[[ 1]
 [-1]]
MATCHED: y matches expected_y!
```

$$V^T = RFx()$$

```
In [21]: VT_3 = get_RFx()
         visualize(VT = VT_3, show_DVT=False, show_UDVT=False)
```



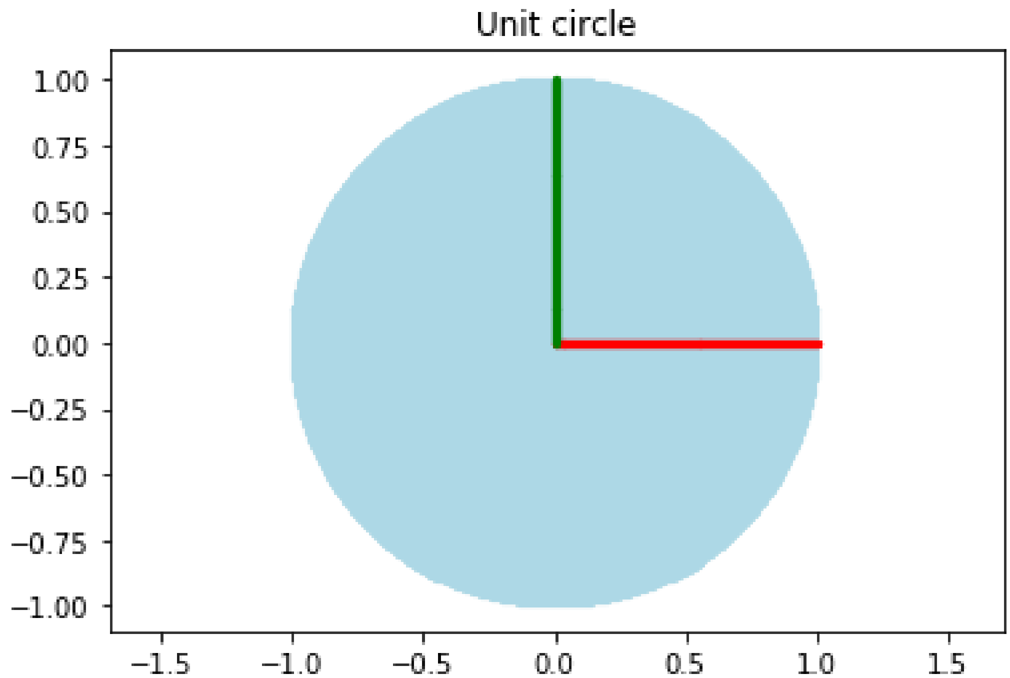


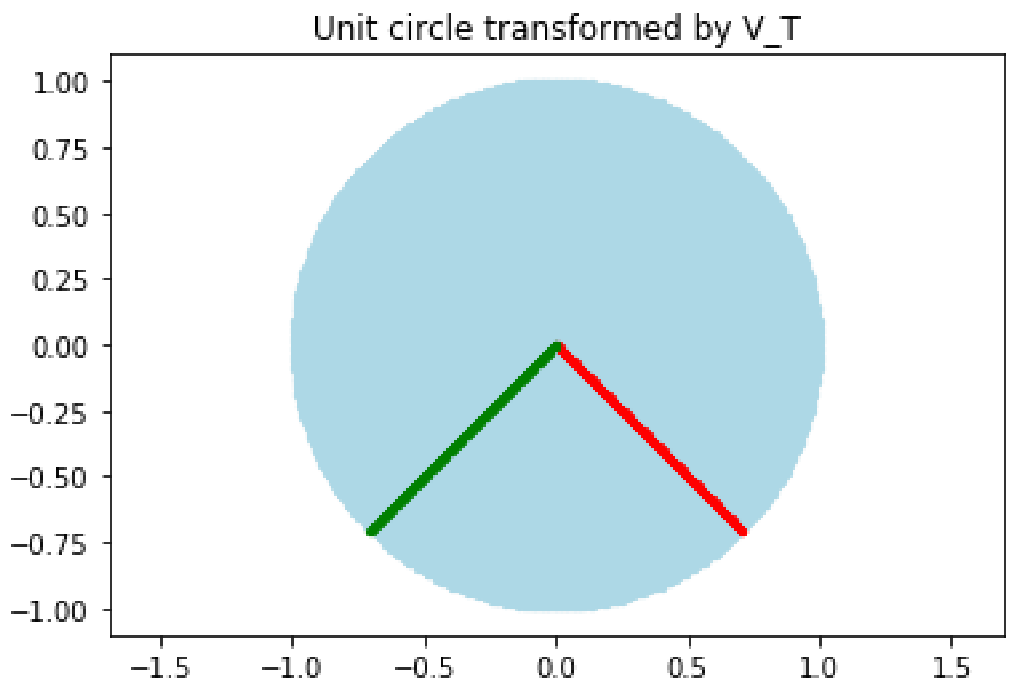
3.7 V^T as a composition of reflection and rotation matrix

In general an orthogonal transformation can be viewed as compositions of rotation and reflection operators Next we observe the effect of setting
$V^T = RFx()RCC\left(\frac{\pi}{4}\right)$

```
In [22]: VT_4 = np.matmul(VT_3, VT_1)
         #Check that VT_4 is still orthogonal
         print("VT_4 is orthogonal?: ", is_orthogonal(VT_4))
         visualize(VT = VT_4, show_DVT=False, show_UDVT=False)
```

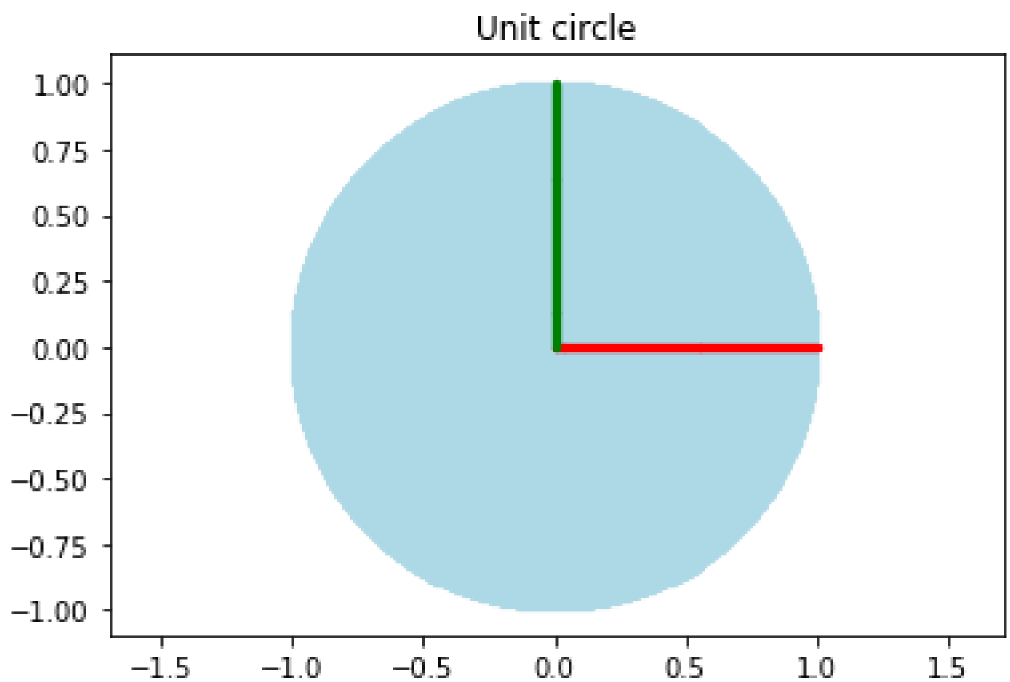
```
VT_4 is orthogonal?:  True
```

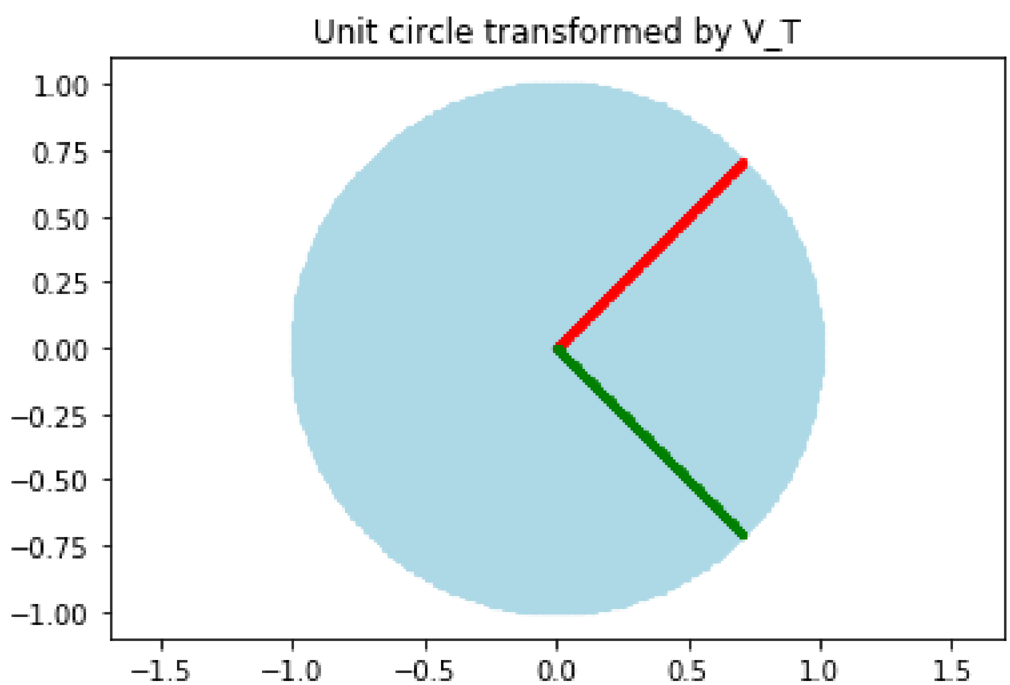

Unit circle
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5


Unit circle transformed by V_T
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

3.8 Q3b iii) Comment on the effect of $V^T = RCC\left(\frac{\pi}{4}\right) RFx()$. Is it same as the case when $V^T = RFx()RCC\left(\frac{\pi}{4}\right)$?

In [23]:
```
VT_5 =  np.matmul(VT_1, VT_3)
visualize(VT = VT_5, show_DVT=False, show_UDVT=False)
```





3.9 #TODO: Fill in solution to Q3b iii here

It is not the case that these rotation matrices commute. In other words for rotation matrices A and B, AB does not equal BA.

4 Effect of linear transformation by diagonal matrix D

The diagonal matrix D with entries σ_1 and σ_2, transforms the unit circle into an ellipse with x direction scaled by σ_1 and y direction scaled by σ_2.

If $\sigma_1 > \sigma_2$, then the major axis of the ellipse will be along the x-axis.

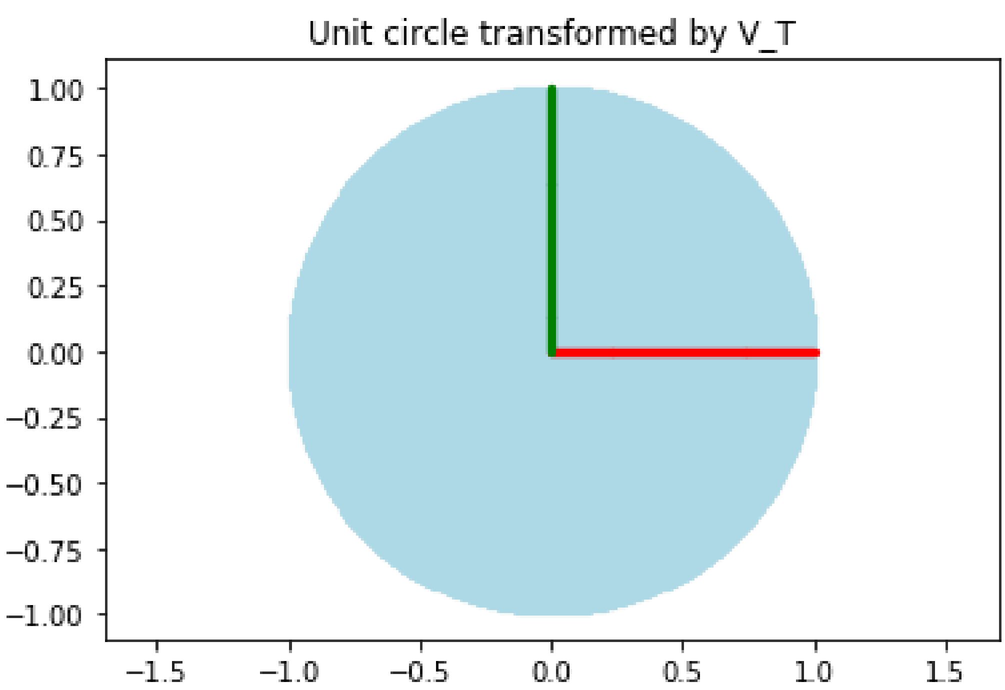
If $\sigma_1 < \sigma_2$, then the major axis of the ellipse will be along the y-axis.

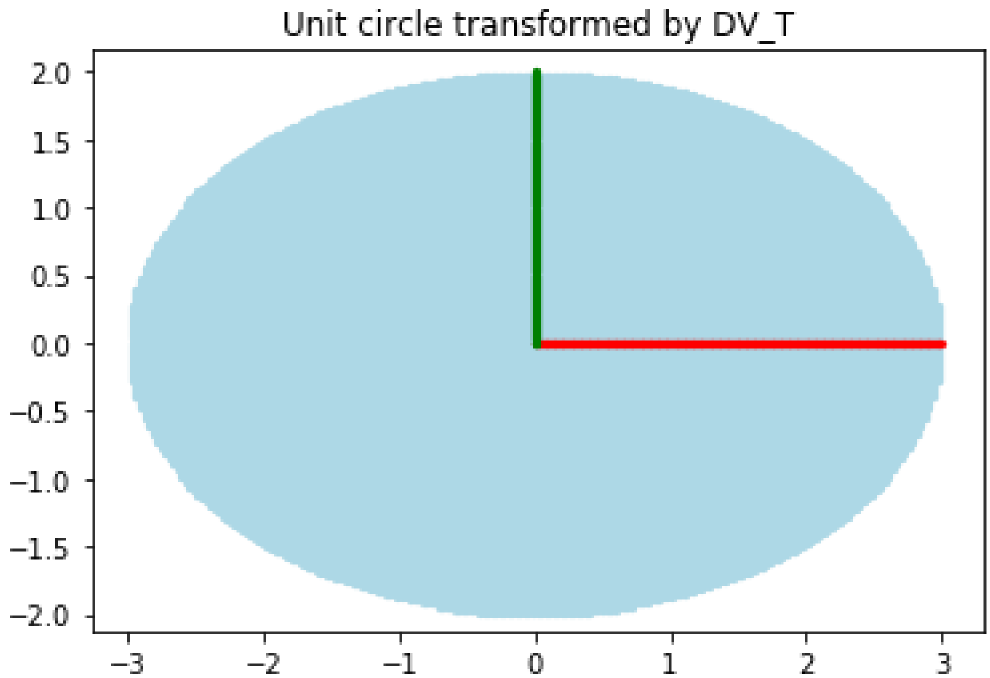
If $\sigma_1 = \sigma_2$, then the ellipse will have both axis equal (i.e it is a circle).

Note that multiplying by D, does not rotate or reflect points in any way. It is a purely scaling operation where different directions get scaled by different values based on entries of D.

4.1 Q3c i: Comment on the length of major and minor axis of the ellipse and their orientation with respect to X and Y axis when D has entries [3, 2].

```
In [24]: D_1 = np.array([3, 2])
         visualize( D = D_1, show_original=False, show_UDVT=False)
```



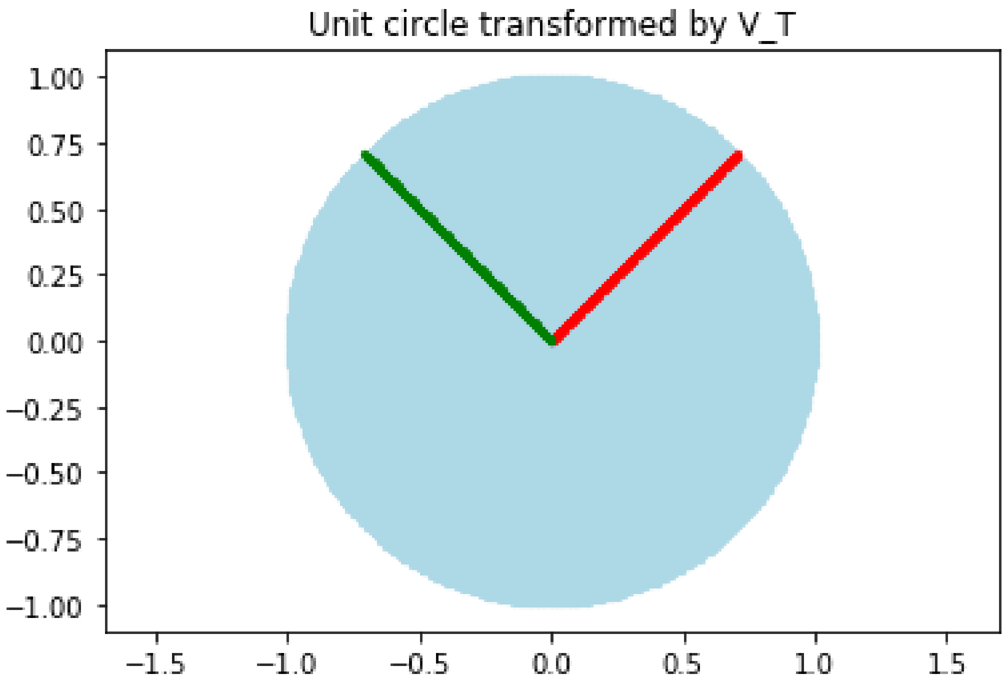


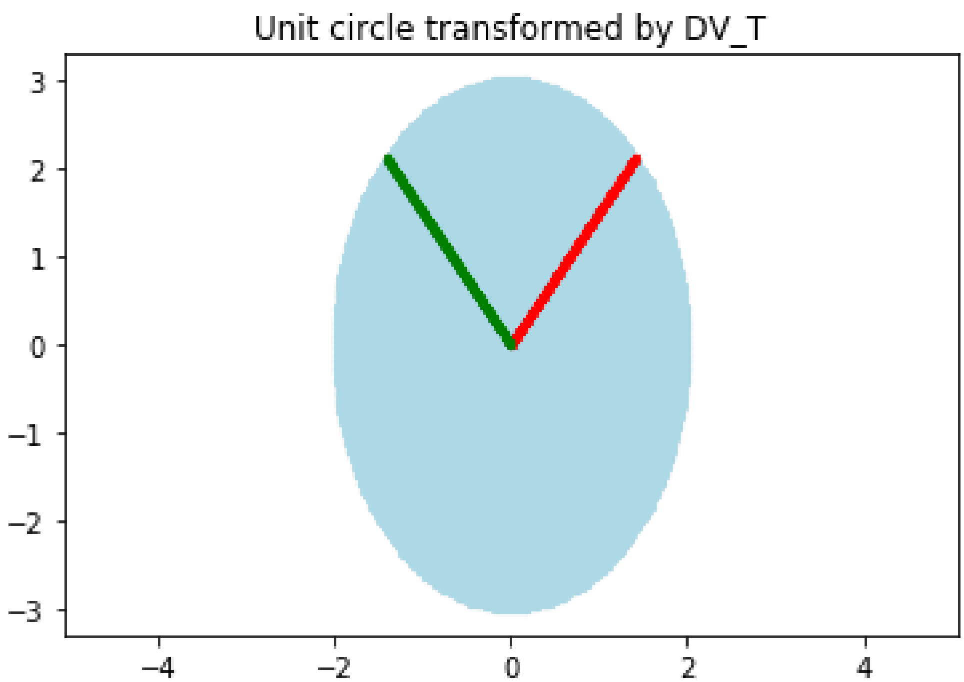
4.2 #TODO: Solution to Q3c i

When D has the entries [3,2] the major axis is the x axis and had been scaled by a factor of 3, while the minor axis is the y axis which has been scaled by a factor of 2.

4.3 Q3c ii: Comment on the length of major and minor axis of the ellipse and their orientation with respect to X and Y axis when D has entries [2, 3].

```
In [25]: D_2 = np.array([2, 3])
         visualize( D = D_2, VT = get_RCC(np.pi/4), show_original=False, show_UDVT=False)
```


Unit circle transformed by V_T
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

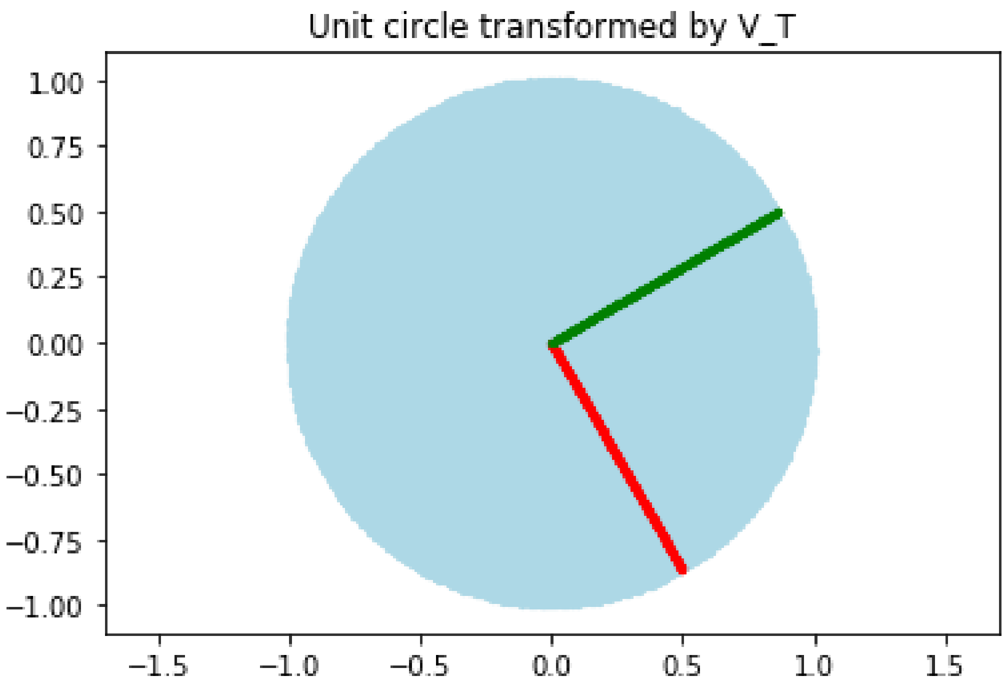

Unit circle transformed by DV_T
3
2
1
0
−1
−2
−3
−4
−2
0
2
4

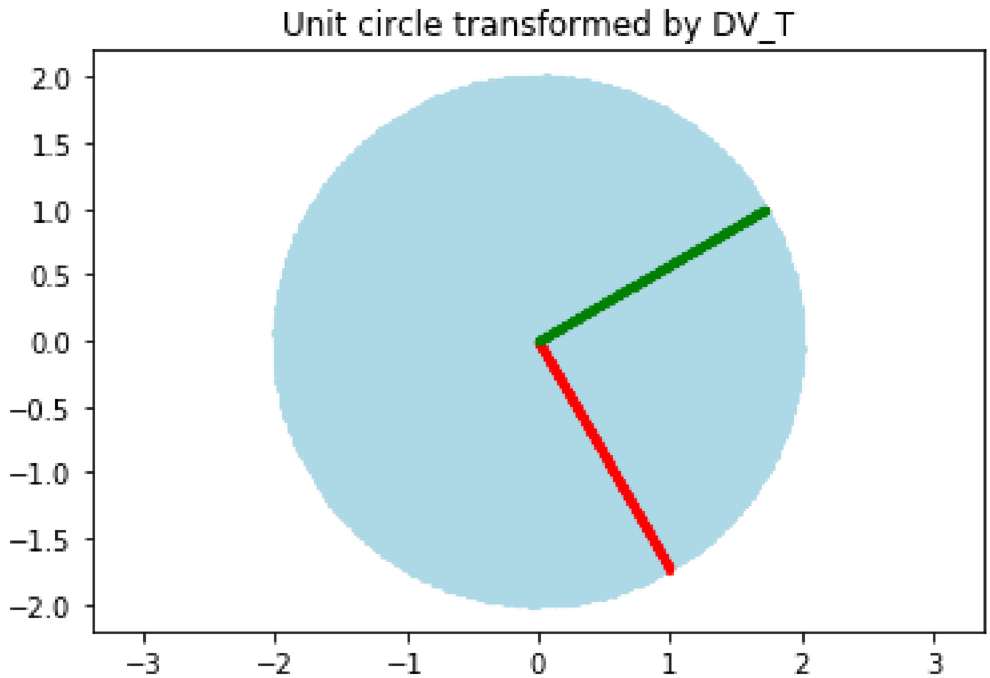
4.4 #TODO Enter solution to Q3c ii here

When D has the entries [2,3] the major axis is the y axis and had been scaled by a factor of 3, while the minor axis is the x axis which has been scaled by a factor of 2.

4.5 Q3c iii: What can you say about the ellipse when D has entries [2, 2]?

In [26]:
```
D_3 = np.array([2, 2])
visualize( D = D_3, VT = get_RCC(-np.pi/3), show_original=False, show_UDVT=False)
```



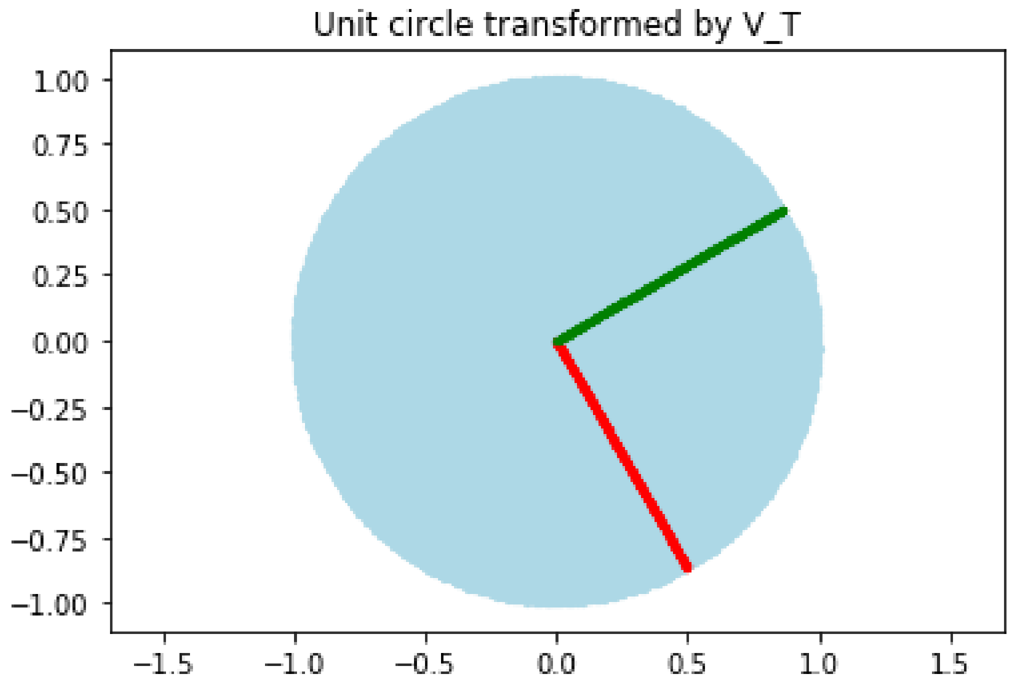


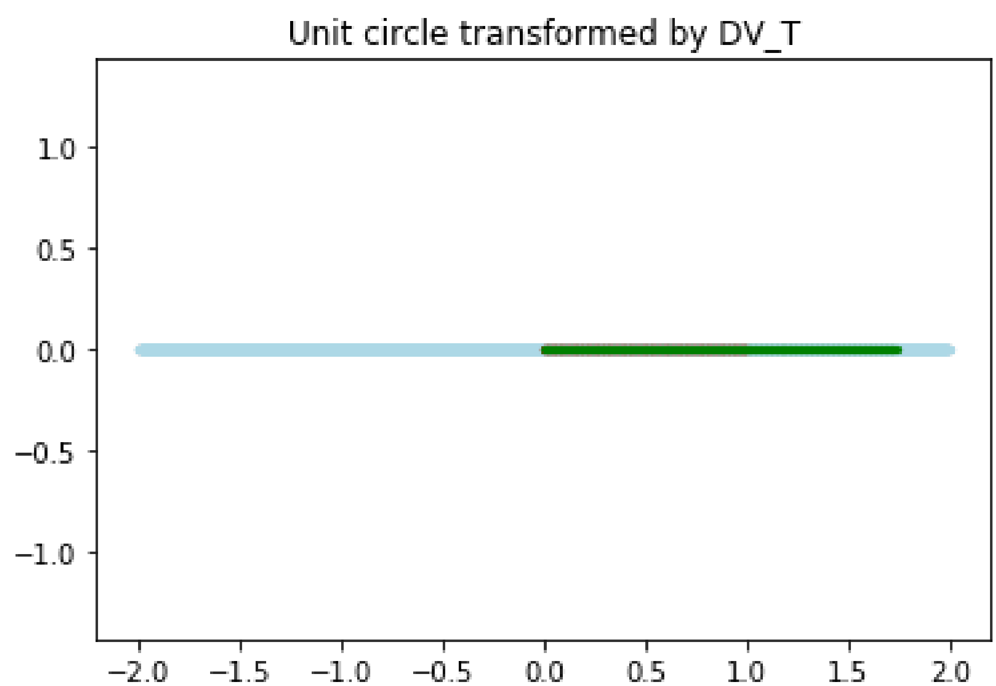
4.6 #TODO Enter solution to Q3c iii here

The circle has been scaled by a factor of two along both axes.

4.7 Q3c iv: What can you say about the ellipse when D has entries [2, 0]?

In [27]:
```
D_4 = np.array([2, 0])
visualize( D = D_4, VT = get_RCC(-np.pi/3), show_original=False, show_UDVT=False)
```


Unit circle transformed by V_T
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5


Unit circle transformed by DV_T
1.0
0.5
0.0
−0.5
−1.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

4.8 #TODO: Enter solution to Q3c iv here

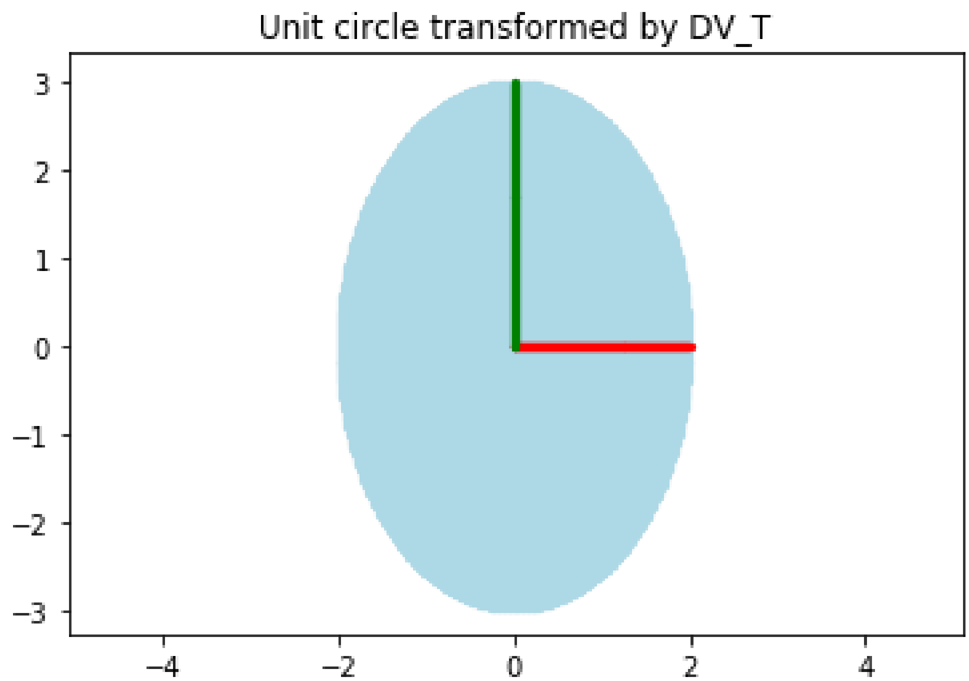
The X axis of the ellipse has been scaled by a factor of 2, while the y component has been gotten rid of (scaled by a factor of 0).

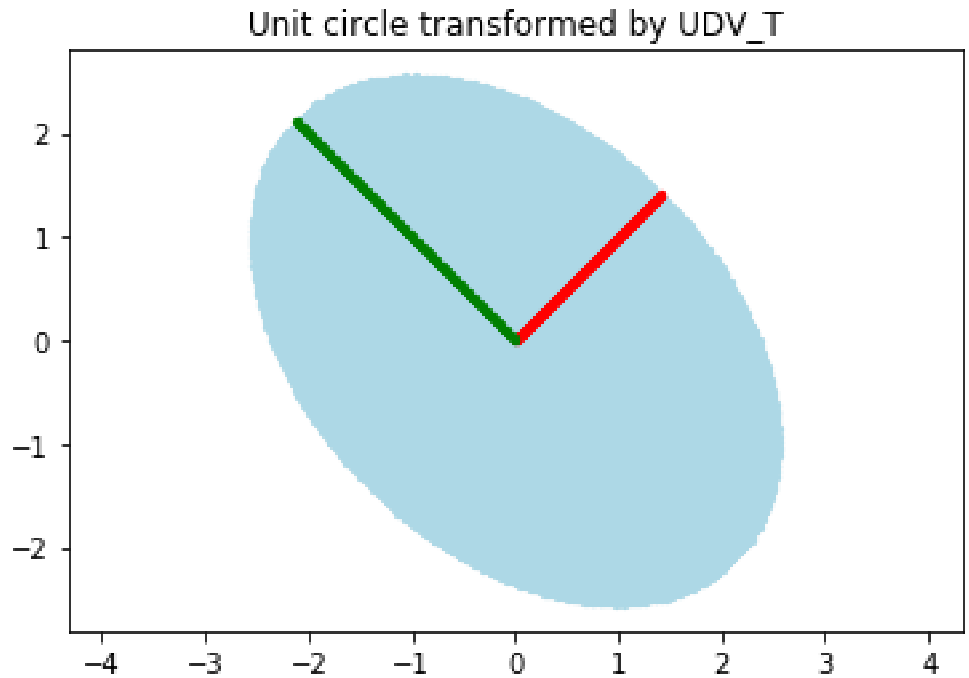
5 Effect of the linear transformation by orthogonal matrix U

As we saw before for V^T, a 2 x 2 orthogonal matrix can be viewed as a linear transformation that performs some combination of rotations and reflections.

5.1 Q3d i Comment on the effect of $U = RCC\left(\frac{\pi}{4}\right)$ as in cell below. What happened to the ellipse? Did length of major and minor axis change?

```
In [29]: U_1 = get_RCC(np.pi/4)
         visualize( U = U_1, D =np.array([2,3]), show_original=False, show_VT=False)
```



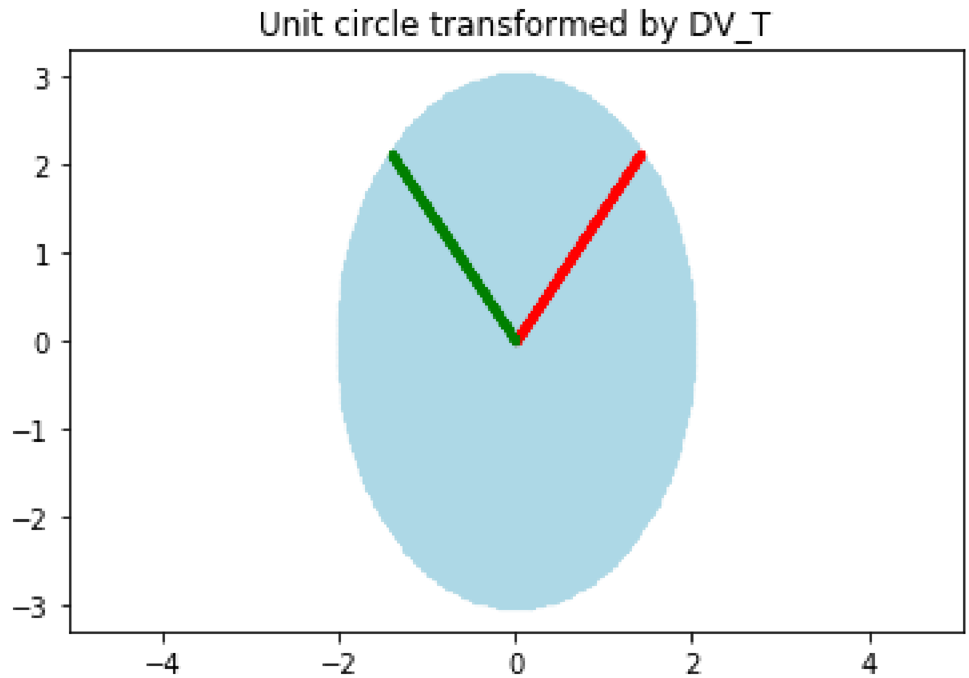


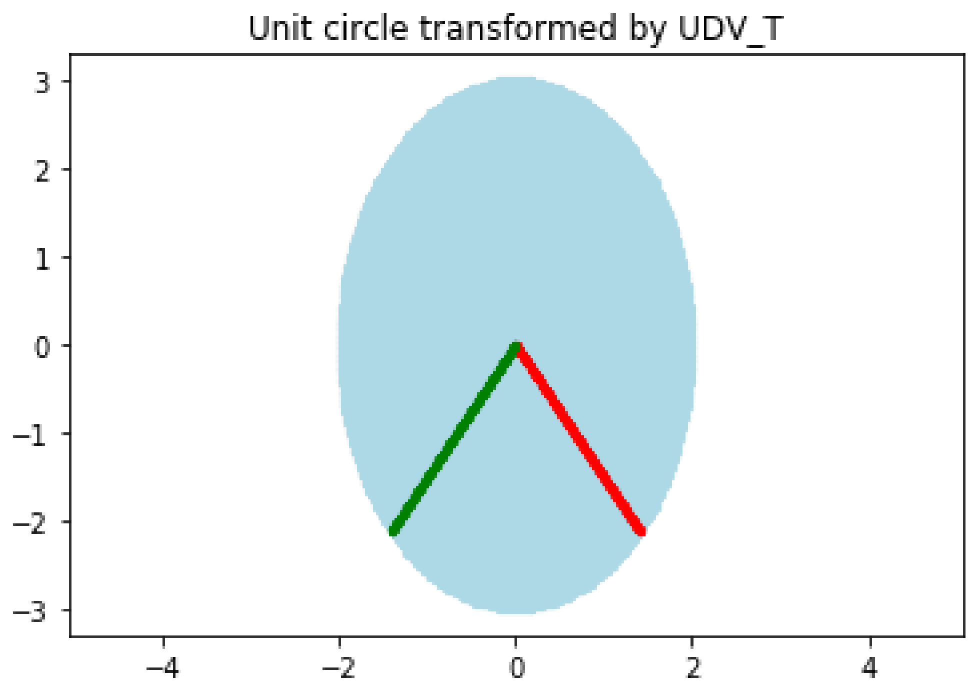
5.2 #TODO: Enter solution to Q3di here

No, the major and minor axes of the ellipse have remained the same length, but was rotated 45 degrees.

5.3 Q3d ii Comment on the effect of $U = RFx()$ as in cell below. What happened to the ellipse? Did length of major and minor axis change?

```
In [30]: U_2 = get_RFx()
         visualize( U = U_2, D =np.array([2,3]), VT = get_RCC(np.pi/4), show_original=False, sl
```


Unit circle transformed by DV_T
3
2
1
0
−1
−2
−3
−4
−2
0
2
4


Unit circle transformed by UDV_T
3
2
1
0
−1
−2
−3
−4
−2
0
2
4

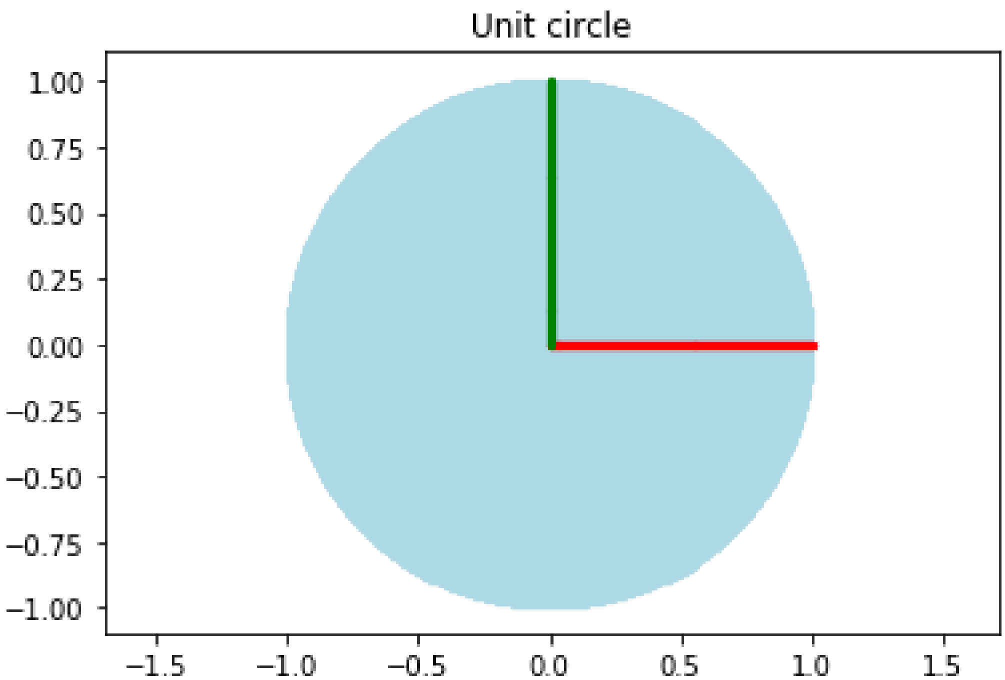
5.4 #TODO Fill in solution to Q3d ii here

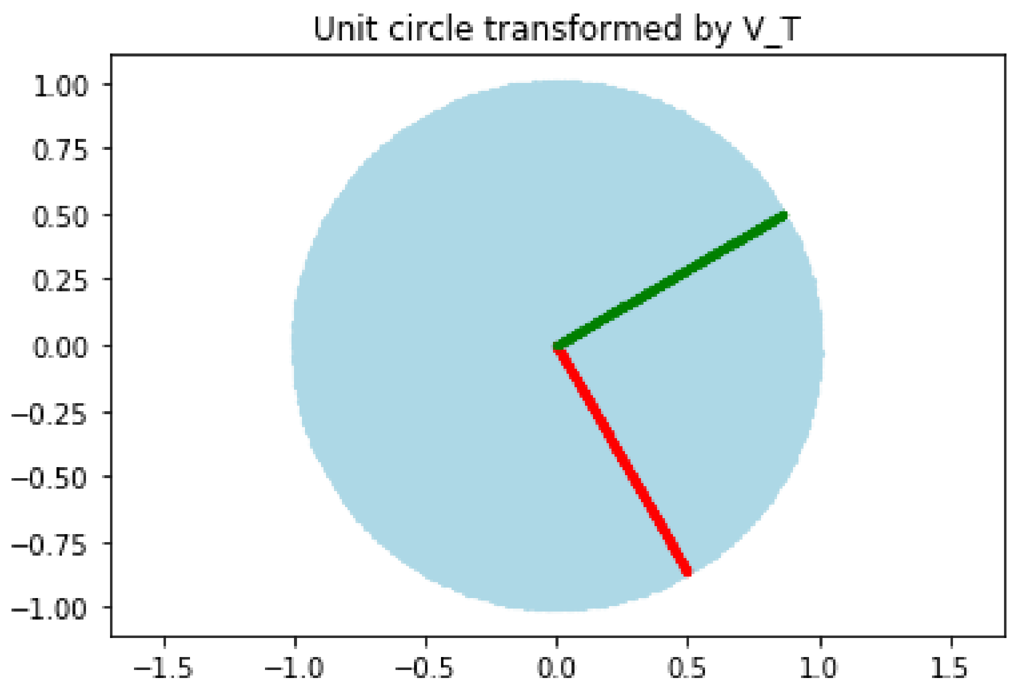
The lengths of the major and minor axes of the elipse have not changed, but the ellipse was reflected across the x axis.

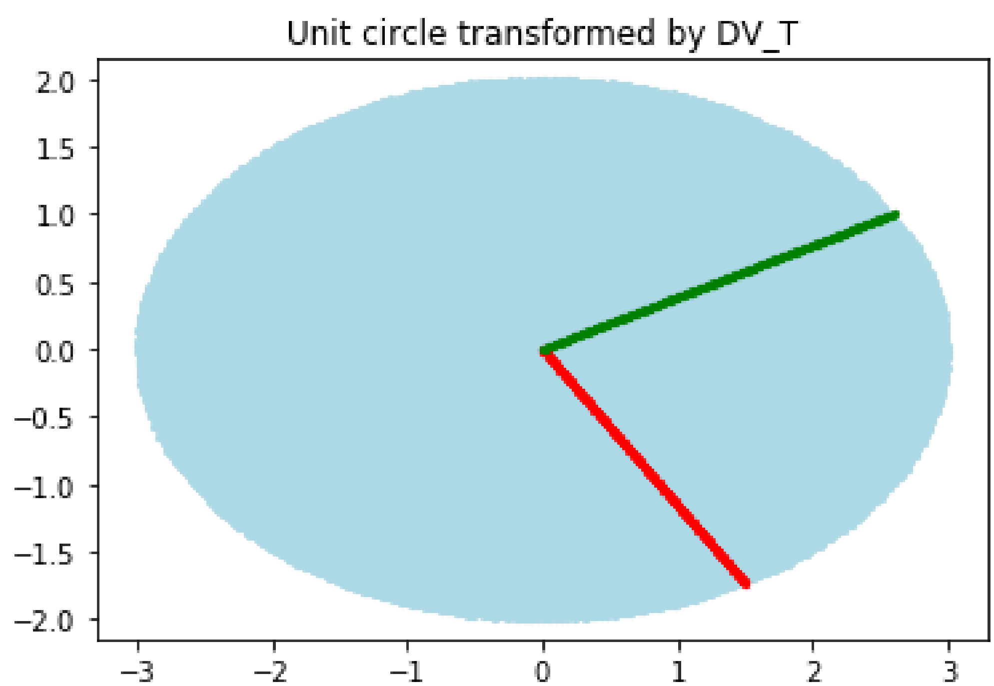
6 Putting everything together. Effect of linear transformation by UDV^T

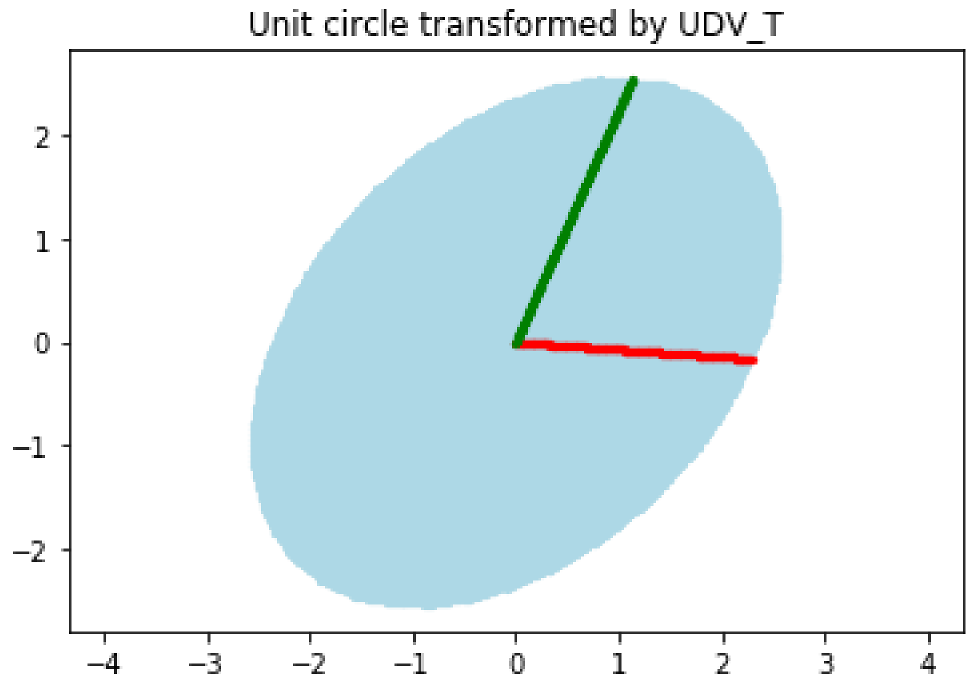
6.0.1 Case I

```
In [32]: U = get_RCC(np.pi/4)
         VT = get_RCC(-np.pi/3)
         D = np.array([3,2])
         visualize(U = U, VT= VT, D=D)
```




Unit circle transformed by V_T
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

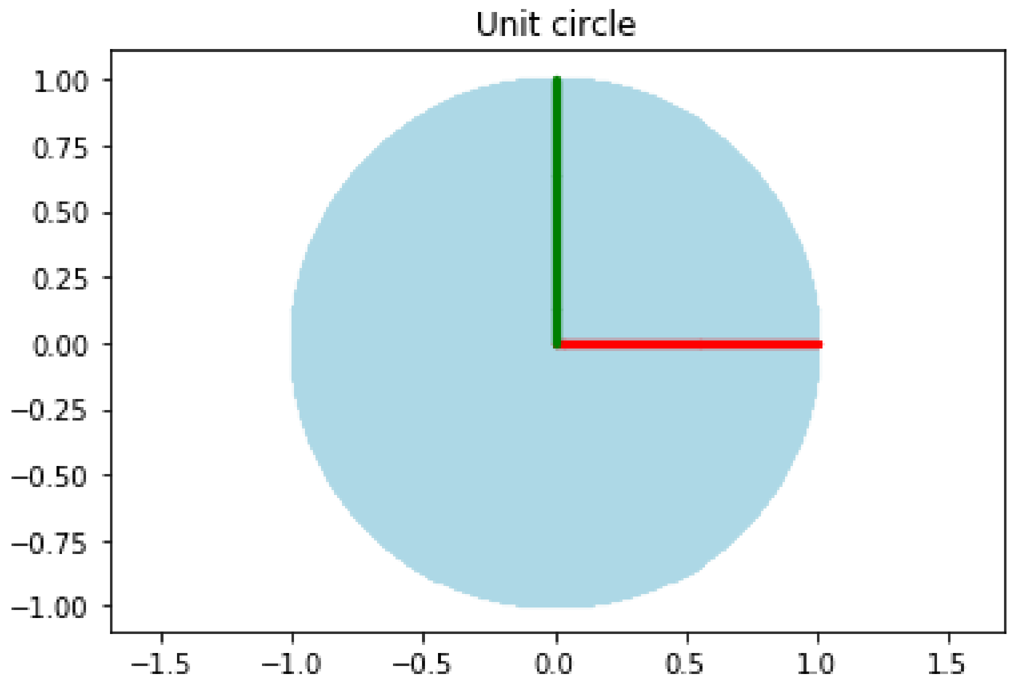

Unit circle transformed by DV_T
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−3
−2
−1
0
1
2
3

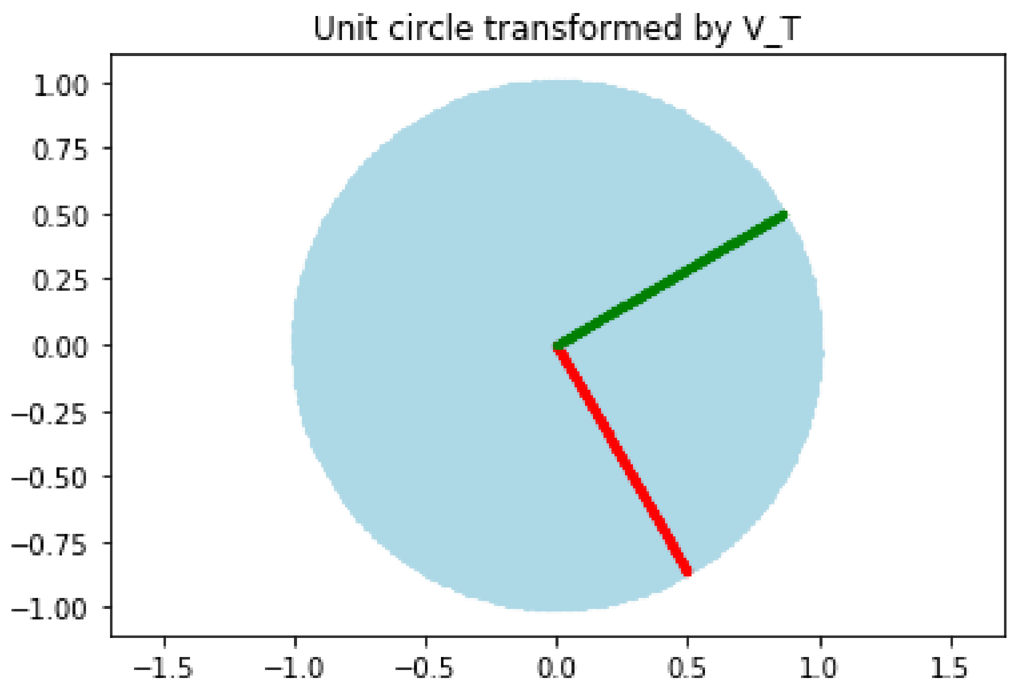


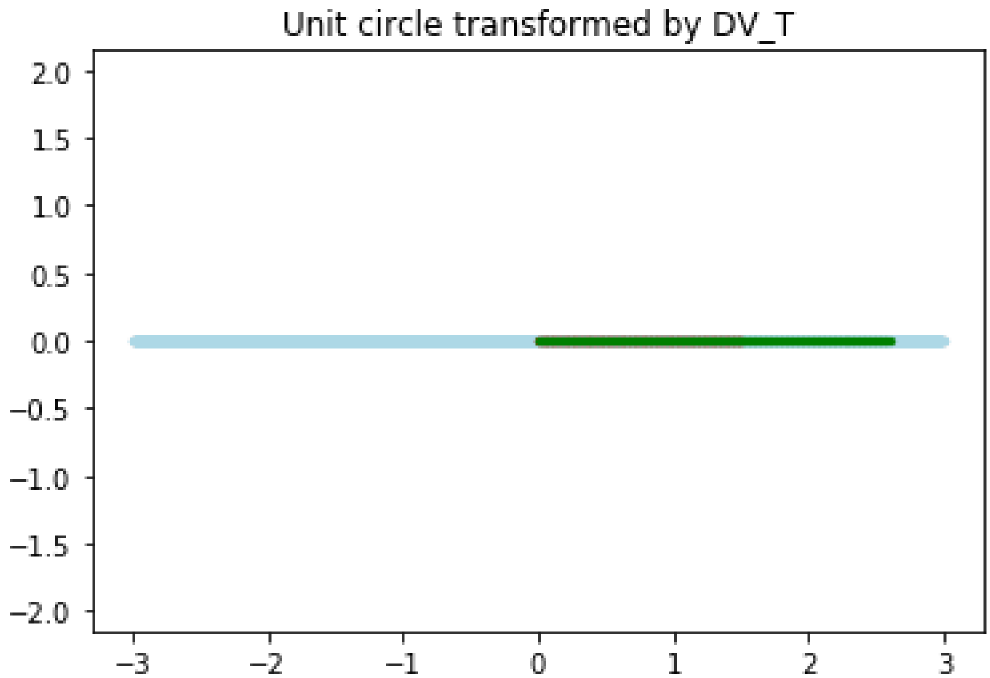
The above figures show the transformation after each step.

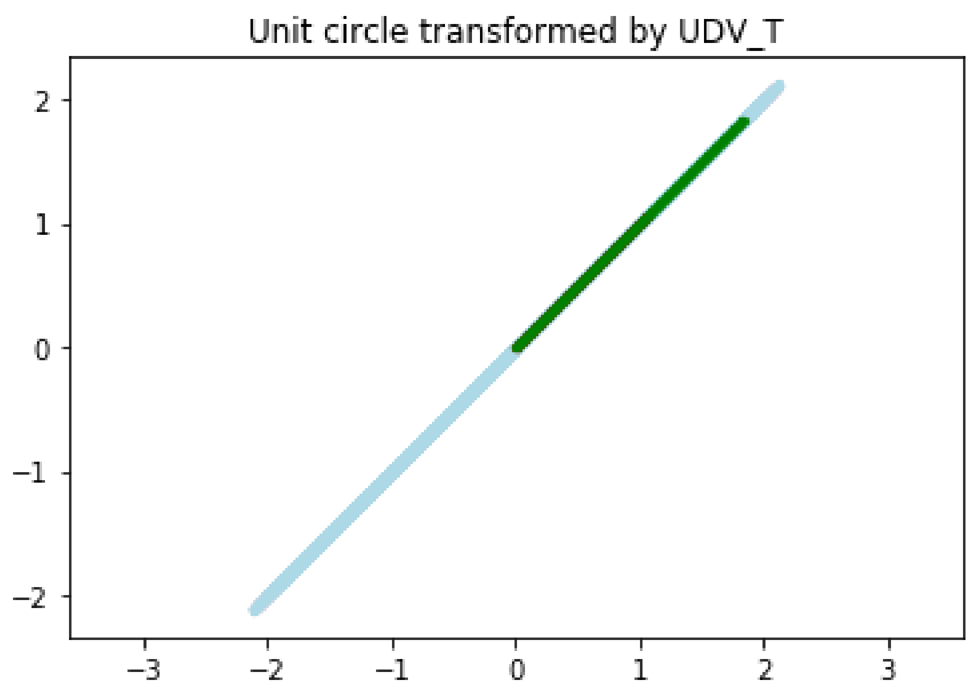
6.0.2 Case II

```
In [33]: U = get_RCC(np.pi/4)
         VT = get_RCC(-np.pi/3)
         D = np.array([3,0])
         visualize(U = U, VT= VT, D=D)
```


Unit circle
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5


Unit circle transformed by V_T
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

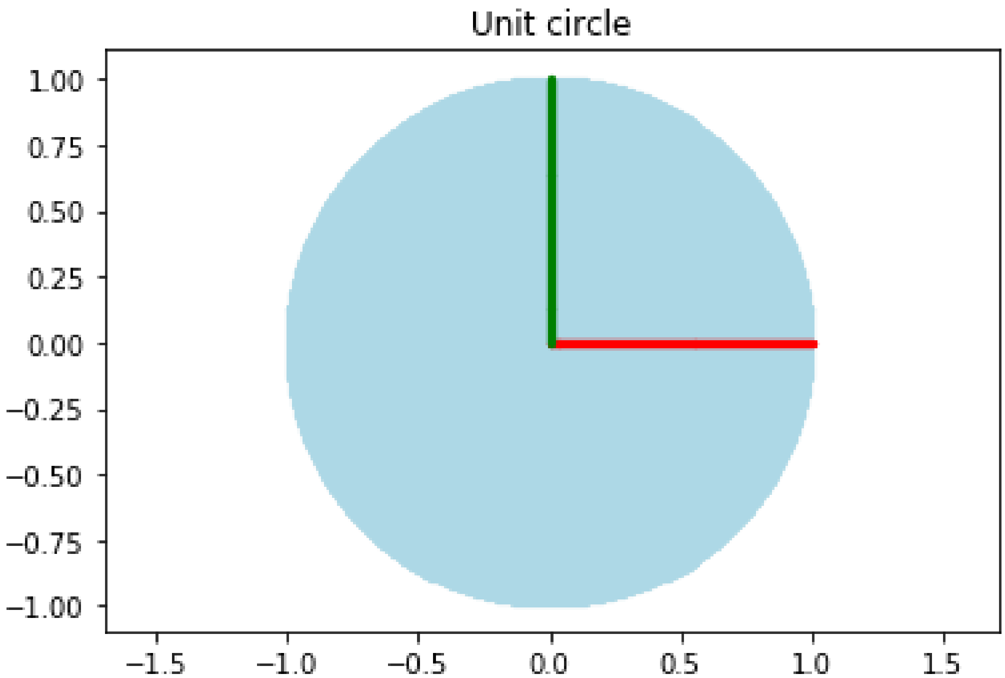


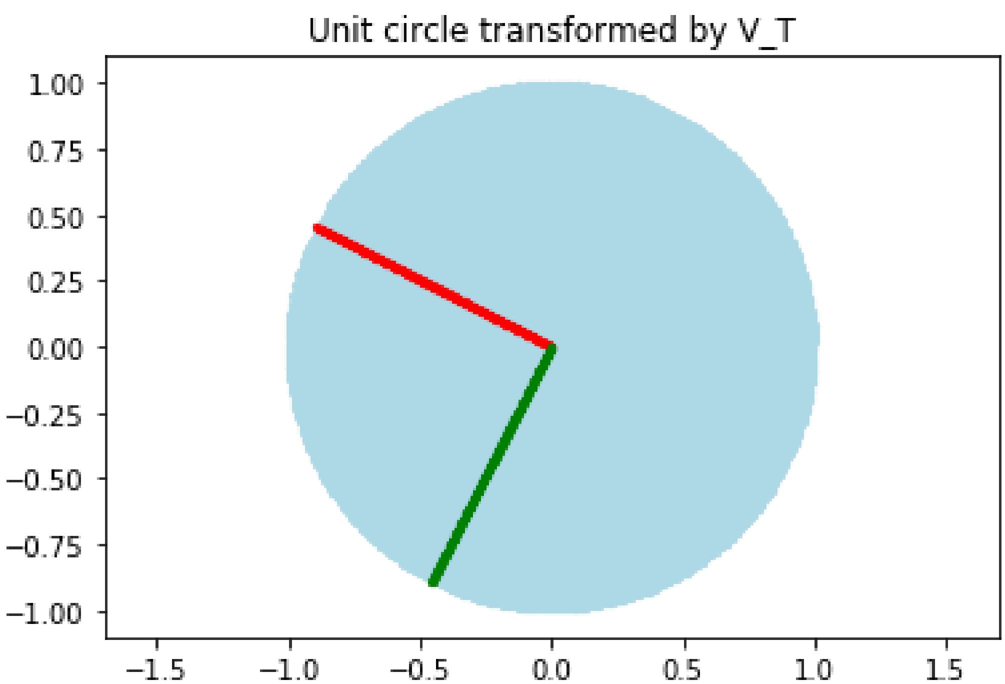


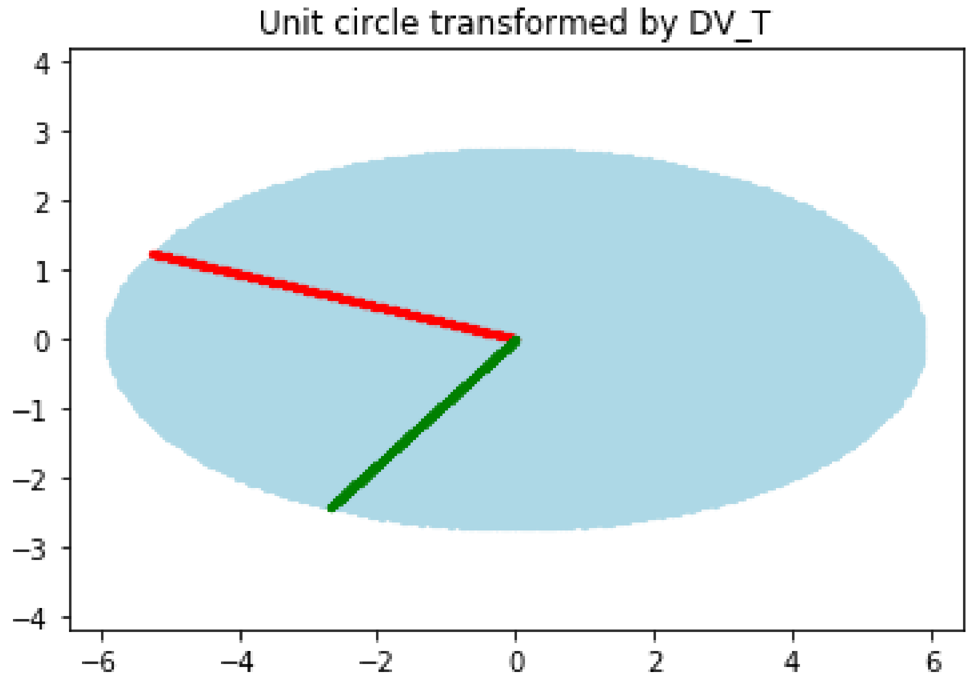
The above figures show the transformation after each step.

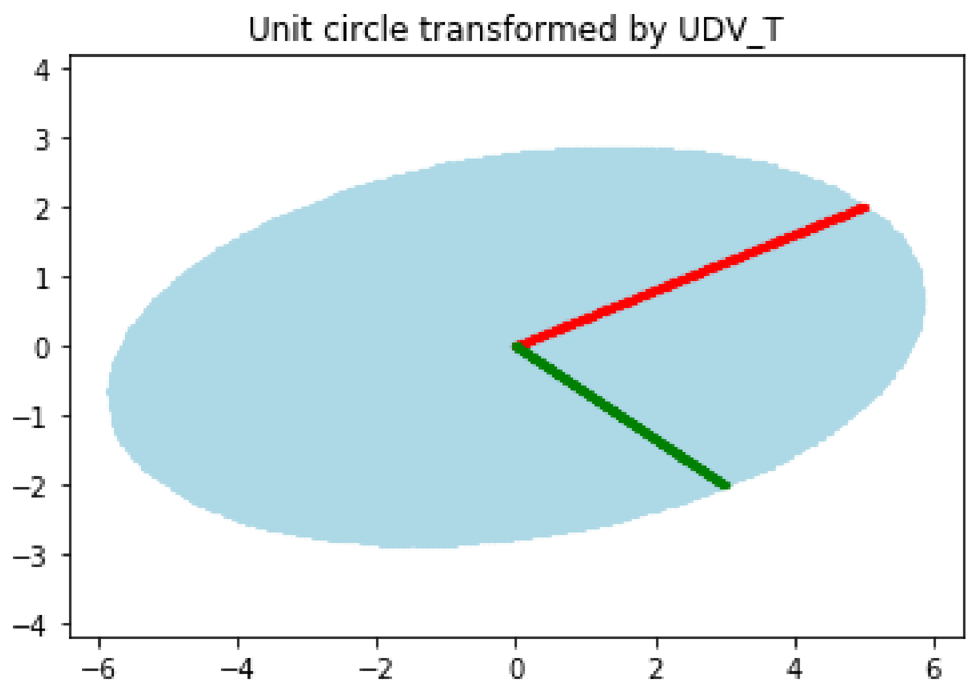
6.0.3 Case III

```
In [34]: A = np.array([[5, 3], [2, -2]])
         U,D,VT = np.linalg.svd(A)
         visualize(U = U, D=D, VT=VT)
```






Unit circle transformed by DV_T
4
3
2
1
0
−1
−2
−3
−4
−6
−4
−2
0
2
4
6


Unit circle transformed by UDV_T
4
3
2
1
0
−1
−2
−3
−4
−6
−4
−2
0
2
4
6

6.1 Q3e For case III, based on the figures obtained by running the cell, answer the following questions:

1) Is V^T a pure rotation, pure reflection or combination of both?
2) Let σ_1 and σ_2 denote the entries of the diagonal matrix in SVD of A, with $\sigma_1 > \sigma_2$? What is an approximate value of $\frac{\sigma_1}{\sigma_2}$?
3) Is U a pure rotation, pure reflection or combination of both?

6.2 #TODO Enter solution to Q3e here

1: V^T is a pure rotation.
2: $\frac{\sigma_1}{\sigma_2}$ is somewhat more than 2 but less that 3.
3: $\hat{U}$ is a combination of a rotation and a reflection.

7 Exploration Area (Not part of homework question)

You are free to visualize the effect of the SVD transformation on the unit circle for whatever matrix you desire

```
In [ ]: # #Sample format 1
        # U = get_RCC(np.pi/4)
        # VT = get_RCC(-np.pi/3)
        # D = np.array([3,2])
        # visualize(U = U, VT= VT, D=D)
```

```
In [ ]: # #Sample format 2
        # A = np.array([[5, 3], [2, -2]])
        # U,D,VT = np.linalg.svd(A)
        # visualize(U = U, D=D, VT=VT)
```

senator_pca_qns

February 21, 2019

0.1 PCA and Senate Voting Data

0.1.1 Places where you have to write code are marked with #TODO

In this problem we are given, X the $m \times n$ data matrix with entries in $\{-1, 0, 1\}$, where each row corresponds to a Senator, and each column to a bill.

```
In [1]: # Import the necessary packages for data manipulation, computation and PCA
        import pandas as pd
        import numpy as np
        import scipy as sp
        import matplotlib.pyplot as plt
        from sklearn.decomposition import PCA
        %matplotlib inline

        np.random.seed(7)
```

```
In [2]: senator_df =  pd.read_csv('senator_data_pca/data_matrix.csv')
        affiliation_file = open("senator_data_pca/politician_labels.txt", "r")
        affiliations = [line.split('\n')[0].split(' ')[1] for line in affiliation_file.readlin
        X = np.array(senator_df.values[:, 3:].T, dtype='float64') #transpose to get senators a
        print("X.shape: ", X.shape)
        n = X.shape[0] #Number of senators
        m = X.shape[1] #Number of bills
```

```
X.shape:  (100, 542)
```

We see that the number of rows n, is the number of senators and is equal to 100. The number of columns, m is the number of bills and is equal to 542.

```
In [3]: typical_row = X[0,:]
        print(typical_row.shape)
        print(typical_row)
```

```
(542,)
[ 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.  0. -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.  0.  0.  1.  1. -1.  1.
  1. -1.  1.  1. -1.  1. -1. -1.  1.  1.  1.  1.  0. -1. -1.  1.  1. -1.
  1.  1.  1.  1.  1.  0.  1.  0.  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.  0.  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.  0.  1.  0. -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.  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.  0.  1. -1.  1.  1.  1.  1.  1. -1. -1. -1.  1.  1.  0.  1.  1.  1.
  1.  1.  1.  1. -1. -1.  0.  0.  0.  0.  0.  0.  0.  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.]
```

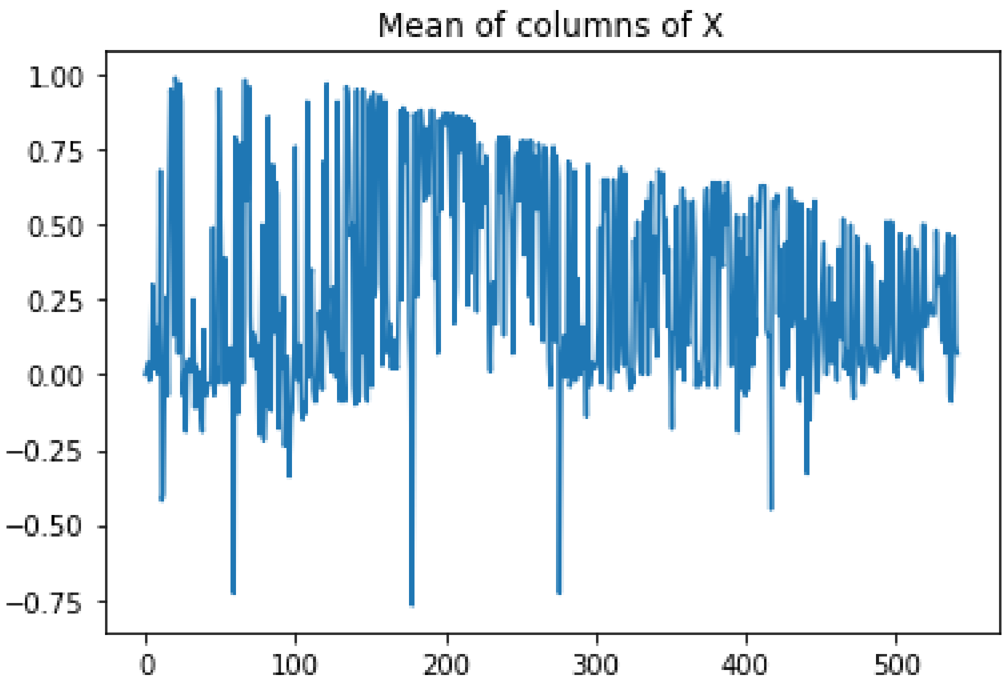
A typical row of X consists of entries -1 (senator voted against), 1(senator voted for) and 0(senator abstained) for each bill.

```
In [4]: typical_column = X[:,0]
        print(typical_column.shape)
        print(typical_column)
```

```
(100,)
[ 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.]
```

A typical row of X consists of entries in {-1, 0, 1} based on how each senator voted for that particular bill.

```
In [5]: X_mean = np.mean(X, axis = 0)
        plt.plot(X_mean)
        plt.title('Mean of columns of X')
        plt.show()
```



We see that the mean of the columns is not zero so we center the data by subtracting the mean

```
In [6]: X_original = X.copy()
        X = X - np.mean(X, axis = 0)
```

0.1.2 Part a) Finding a unit-norm m-vector a to maximize variance

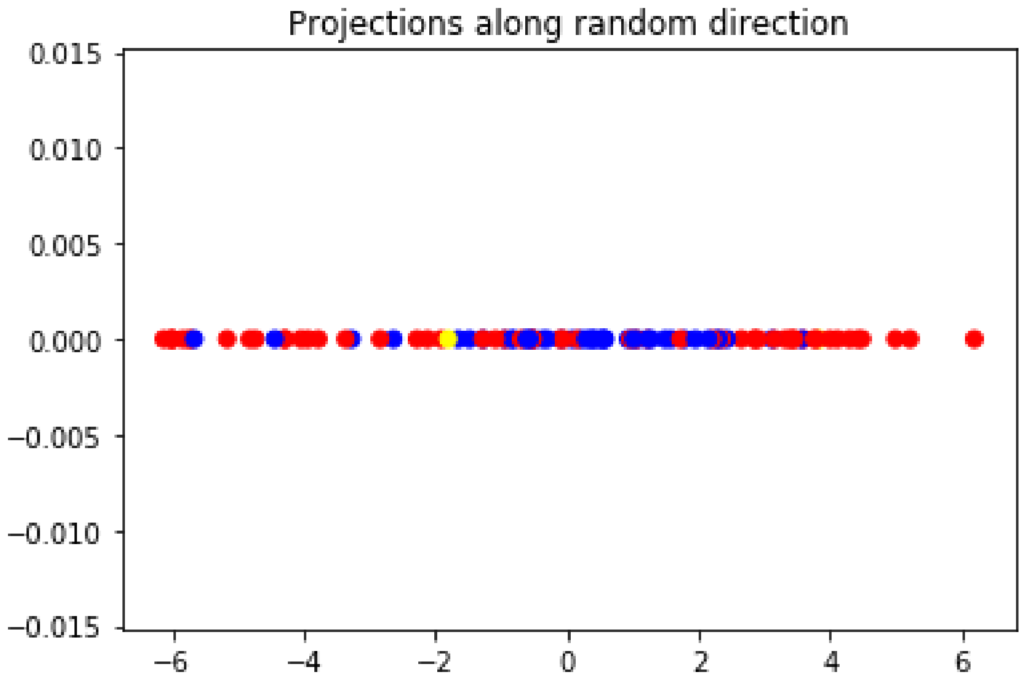
This is a function to calculate the scores, $f(X, a)$.

```
In [7]: def f(X, a):
            return np.matmul(X,a)
```

Before we calculate the a that maximizes variance, let us observe how the scalar projections on a random direction a look like.

```
In [8]: a_rand = np.random.rand(542,1) #generate a random direction
        a_rand = a_rand/np.linalg.norm(a_rand) #we normalize the vector
        scores_rand = f(X, a_rand) #recall definition of f above
        # Now we visualize the scores along a_rand
```

```
plt.scatter(scores_rand, np.zeros_like(scores_rand), c=affiliations)
plt.title('Projections along random direction')
plt.show()

print("Variance along random direction: ", scores_rand.var())
```



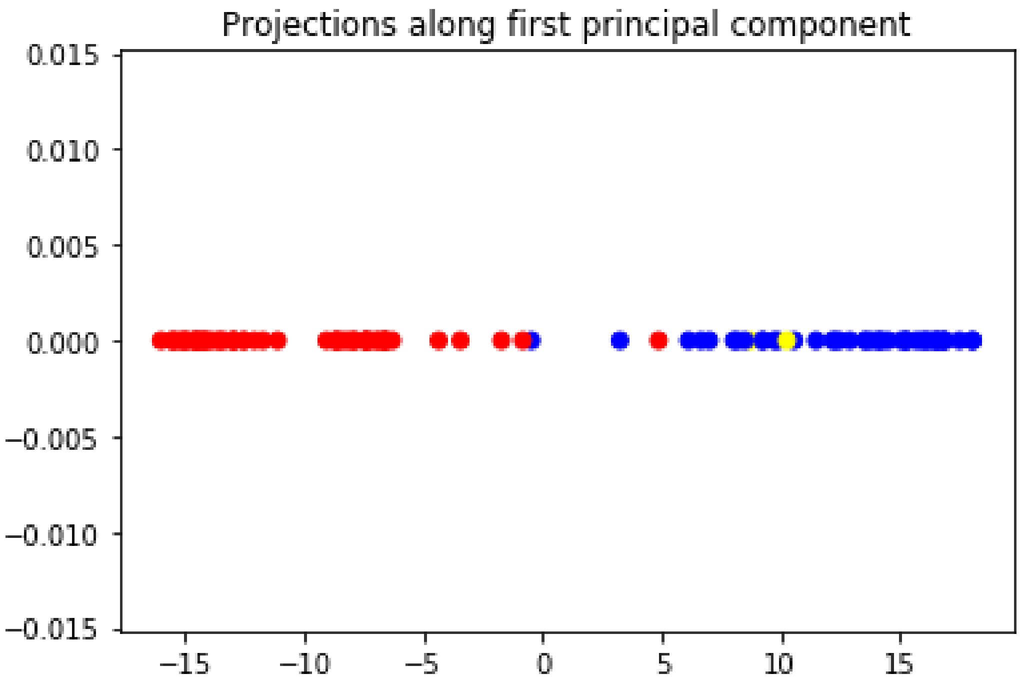

```
Variance along random direction:  9.267454390893336
```

Note here that projecting along the random vector *a_rand* does not explain much variance at all! It is clear that this direction does not give us any information about the senators' affiliations.

Next let us find direction a_1 that maximizes variance. This will be the first principal component of X.

```
In [11]: #TODO: write code to get a_1, the first principal component of X(Note that shape of a_
         pca = PCA(n_components = 1)
         pca.fit(X)
         a_1 = pca.components_[0]
         #TODO replace this line with code to get a_1 that maximizes variance
         #Hint: the PCA packagle imported from sklearn.decomposition will be useful here. Look
         #pca.fit() from its documentation
```

```
#Next we compute scores along first principal component
scores_a_1 = f(X, a_1) #recall definition of f above
plt.scatter(scores_a_1, np.zeros_like(scores_a_1), c=affiliations)
plt.title('Projections along first principal component')
plt.show()

print("Variance along first principal component: ", scores_a_1.var())
```




```
Variance along first principal component:  149.7489650762074
```

We can see that majority of the blue is close to one side of the axis and red is close to the other side. This shows that the first principal component direction explains the vote spread tends to align with party affiliation.

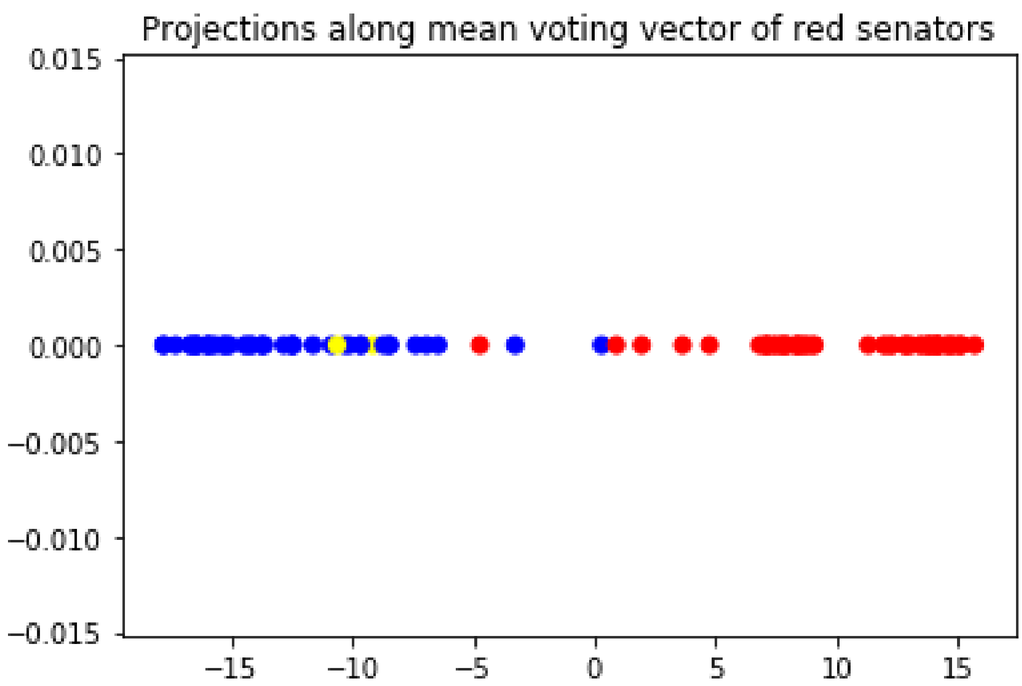
0.1.3 Part b) Comparison to party averages

Building on the observation that senators vote in line with their party average let us compute variance along the following two directions: a_mean_red: Unit vector along mean of rows corresponding to RED senators
a_mean_blue: Unit vector along mean of rows corresponding to BLUE senators

```
In [12]: red_X = []
         for i in range(len(affiliations)):
```

```
    if affiliations[i] == 'Red':
        red_X.append(X[i])
print(len(red_X[2].shape))
mu_red = np.mean(red_X, axis = 0) #TODO Replace this line with mu_red (with shape (54
#corresponding to Red senators as given by affiliations.
#Hint: Print out affiliations and check what its entries are:
# print(len(affiliations))
# print(affiliations)

a_mean_red = mu_red/np.linalg.norm(mu_red) # normalize the vector
scores_mean_red = f( X, a_mean_red)
plt.scatter(scores_mean_red, np.zeros_like(scores_mean_red), c=affiliations)
plt.title('Projections along mean voting vector of red senators')
plt.show()

print("Variance along mean voting vector of red senators: ", scores_mean_red.var())

#Let us check angle between this and the first prinicpal component
dot_product_red_a1 = float(np.dot(a_mean_red.T, a_1))
angle_red_a1 = np.arccos(dot_product_red_a1)*180/np.pi

print("Dot product of a_mean_red and a_1:", dot_product_red_a1)
print("Angle between a_mean_red and a_1 in degrees:", angle_red_a1)
```

1



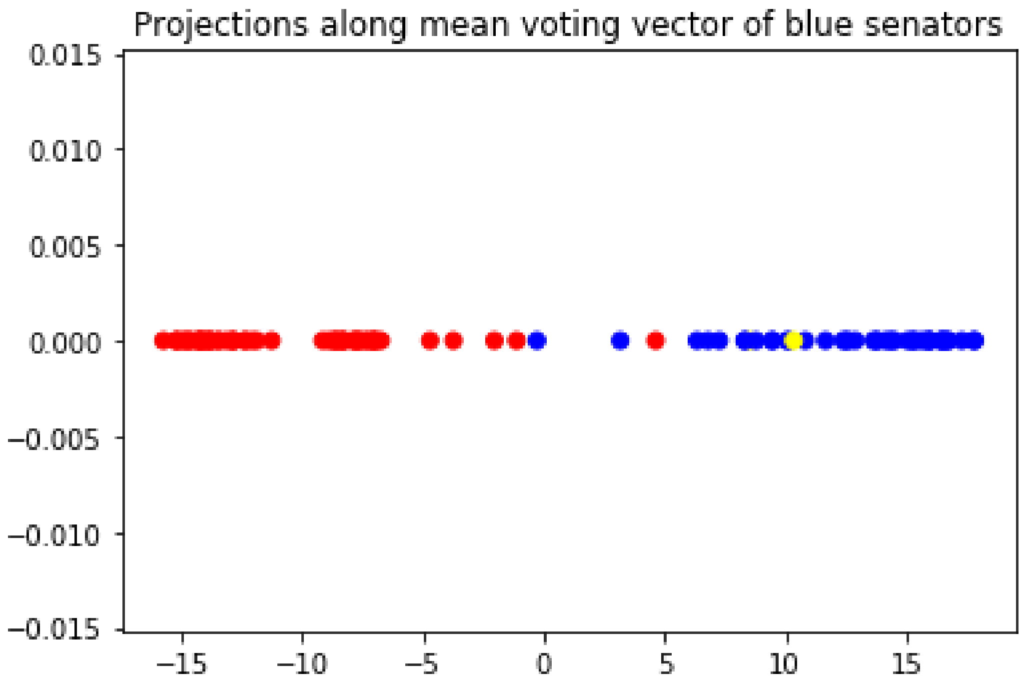
```
Variance along mean voting vector of red senators:  148.80699963205723
Dot product of a_mean_red and a_1: -0.9965356912812968
Angle between a_mean_red and a_1 in degrees: 175.22941782780208
```

In [13]:
```python
red_X[0].shape
```

Out[13]:
```
(542,)
```

In [14]:
```python
blue_X = []
for i in range(len(affiliations)):
    if affiliations[i] == 'Blue':
        blue_X.append(X[i])

mu_blue = np.mean(blue_X, axis = 0) #TODO Replace this line with mu_red (with shape (
#corresponding to Blue senators as given by affiliations.
#Hint: Print out affiliations and check what its entries are:
# print(len(affiliations))
# print(affiliations)

print(mu_blue.shape)

a_mean_blue = mu_blue/np.linalg.norm(mu_blue) # normalize the vector
scores_mean_blue = f( X, a_mean_blue)
plt.scatter(scores_mean_blue, np.zeros_like(scores_mean_blue), c=affiliations)
plt.title('Projections along mean voting vector of blue senators')
plt.show()

print("Variance along mean voting vector of blue senators: ", scores_mean_blue.var())

#Let us check angle between this and the first prinicpal component
dot_product_blue_a1 = float(np.dot(a_mean_blue.T, a_1))
angle_blue_a1 = np.arccos(dot_product_blue_a1)*180/np.pi

print("Dot product of a_mean_blue and a_1:", dot_product_blue_a1)
print("Angle between a_mean_blue and a_1 in degrees:", angle_blue_a1)
```

```
(542,)
```




```
Variance along mean voting vector of blue senators:  148.90884144004613
Dot product of a_mean_blue and a_1: 0.9969831227823034
Angle between a_mean_blue and a_1 in degrees: 4.451697983373453
```

```
In [15]: #Finally let us compute angle between a_mean_red and a_mean_blue:
         dot_product_blue_red = float(np.dot(a_mean_blue.T, a_mean_red))
         angle_blue_red = np.arccos(dot_product_blue_red)*180/np.pi

         print("Dot product of a_mean_blue and mean_red:", dot_product_blue_red)
         print("Angle between a_mean_blue and mean_red in degrees:", angle_blue_red)
```

```
Dot product of a_mean_blue and mean_red: -0.9992350984093124
Angle between a_mean_blue and mean_red in degrees: 177.75886458298294
```

0.1.4 #TODO Fill in code to obtain mu_red, and mu_blue in the cells above. Comment on your observations about how the party averages (a_mean_red and a_mean_blue) are related to the first principal component (a_1).

Based on what we see above, a very large portion of the variance can be explained along the direction of the first principal component.

0.1.5 Part c) Computing total variance. Fill in code in cell below to obtain total variance along first two principal components. (Refer to the latex file for more details on the question).

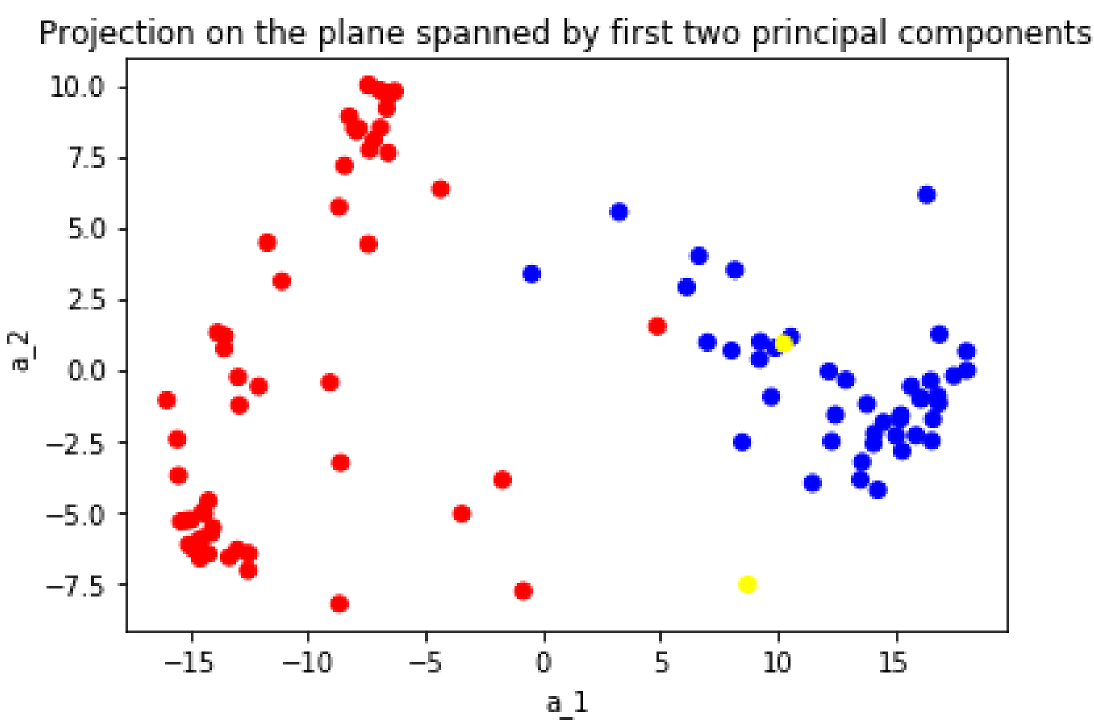
```
In [16]: X_bar = np.matmul(X.T, X)/n
         eigs = np.linalg.eigvals(X_bar)
         total_variance =  eigs[0] + eigs[1]#Replace with correct answer, the sum of largest t

         print("Total variance explained by first two principal components: ", total_variance)
```

```
Total variance explained by first two principal components:  (175.1711616052319+0j)
```

Next we find the projection onto the plane spanned by the first two principal components

```
In [17]: pca = PCA(n_components=2)
         projected = pca.fit_transform(X)

         print(projected.shape)
         plt.scatter(projected[:, 0], projected[:, 1], c=affiliations)
         plt.xlabel('a_1')
         plt.ylabel('a_2')
         plt.title('Projection on the plane spanned by first two principal components')
         plt.show()
```

```
(100, 2)
```



0.2 Part d) Finding bills that are the most/least contentious

0.2.1 Approach 1: Finding variance of columns of X. Note that the variance of column j can be viewed as the variance of scores along the direction e_j, where e_j is a basis vector with one in the jth entry and zero elsewhere.

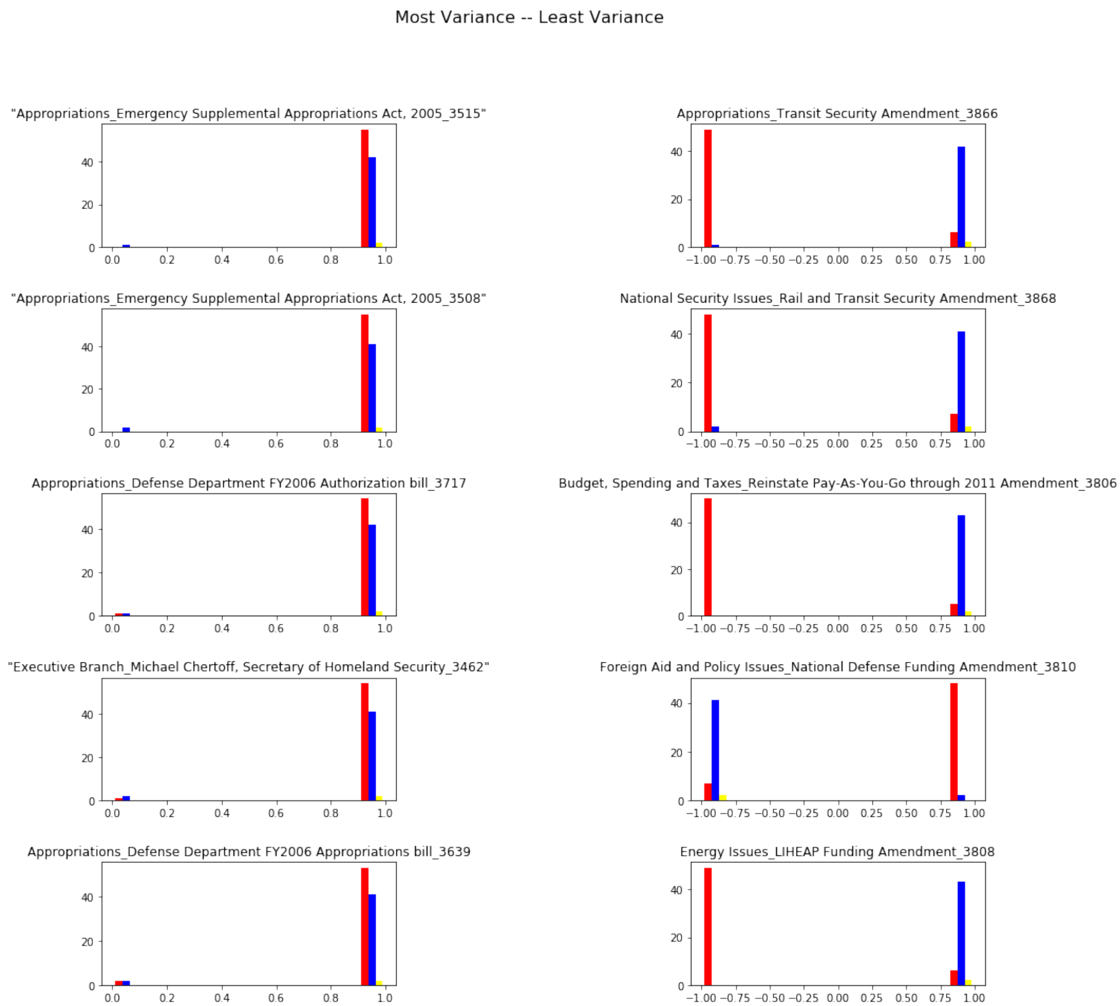
In [19]:
```
list_variances = X.var(0) # projects the standard basis in R^n for all bills; returns
bills = senator_df['bill_type bill_name bill_ID'].values

sorted_idx_variances = np.argsort(list_variances) #TODO remove this line and replace
#compute sorted_idx_variances: a np.array of shape (542,) containing integer entries
# corresponding to decreasing order of variance of scores in list_variances. Hint: Us
#Eg. If list_variances = [1,3,2,4], then sorted_idx_variances should be np.array([3,1



print(sorted_idx_variances.shape)
```

```
(542,)
```

0.2.2 #TODO: Part d i) Fill in code to compute sorted_idx_variances in the cell above

In [20]:
```
# Retrive the bills with the top 5 variances and the lowest 5 variances
top_5 = [bills[sorted_idx_variances[i]] for i in range(5)]
# print(top_10)
bot_5 = [bills[sorted_idx_variances[-1-i]] for i in range(5)]

# print(bot_10)
#We look at voting pattern for bills with most and least variance using original non-
fig, axes = plt.subplots(5,2, figsize=(15,15)) # 1 plot to make things easier to see
for i in range(5):
    idx = sorted_idx_variances[i]

    X_red_c = X_original[np.array(affiliations) == 'Red',idx]
    X_blue_c = X_original[np.array(affiliations) == 'Blue',idx]
    X_yellow_c = X_original[np.array(affiliations) == 'Yellow',idx]

    axes[i,0].hist([X_red_c, X_blue_c, X_yellow_c], color = ['red', 'blue', 'yellow']
    axes[i,0].set_title(bills[idx])


for i in range(1,6):
    idx2 = sorted_idx_variances[-i]
    X_red_c2 = X_original[np.array(affiliations) == 'Red',idx2]
    X_blue_c2 = X_original[np.array(affiliations) == 'Blue',idx2]
    X_yellow_c2 = X_original[np.array(affiliations) == 'Yellow',idx2]

    axes[i-1,1].hist([X_red_c2, X_blue_c2, X_yellow_c2], color = ['red', 'blue', 'yel
```

```
        axes[i-1,1].set_title(bills[idx2])

plt.subplots_adjust(hspace=0.5, wspace = 1)
fig.suptitle('Most Variance -- Least Variance', fontsize=16)
plt.show()
```



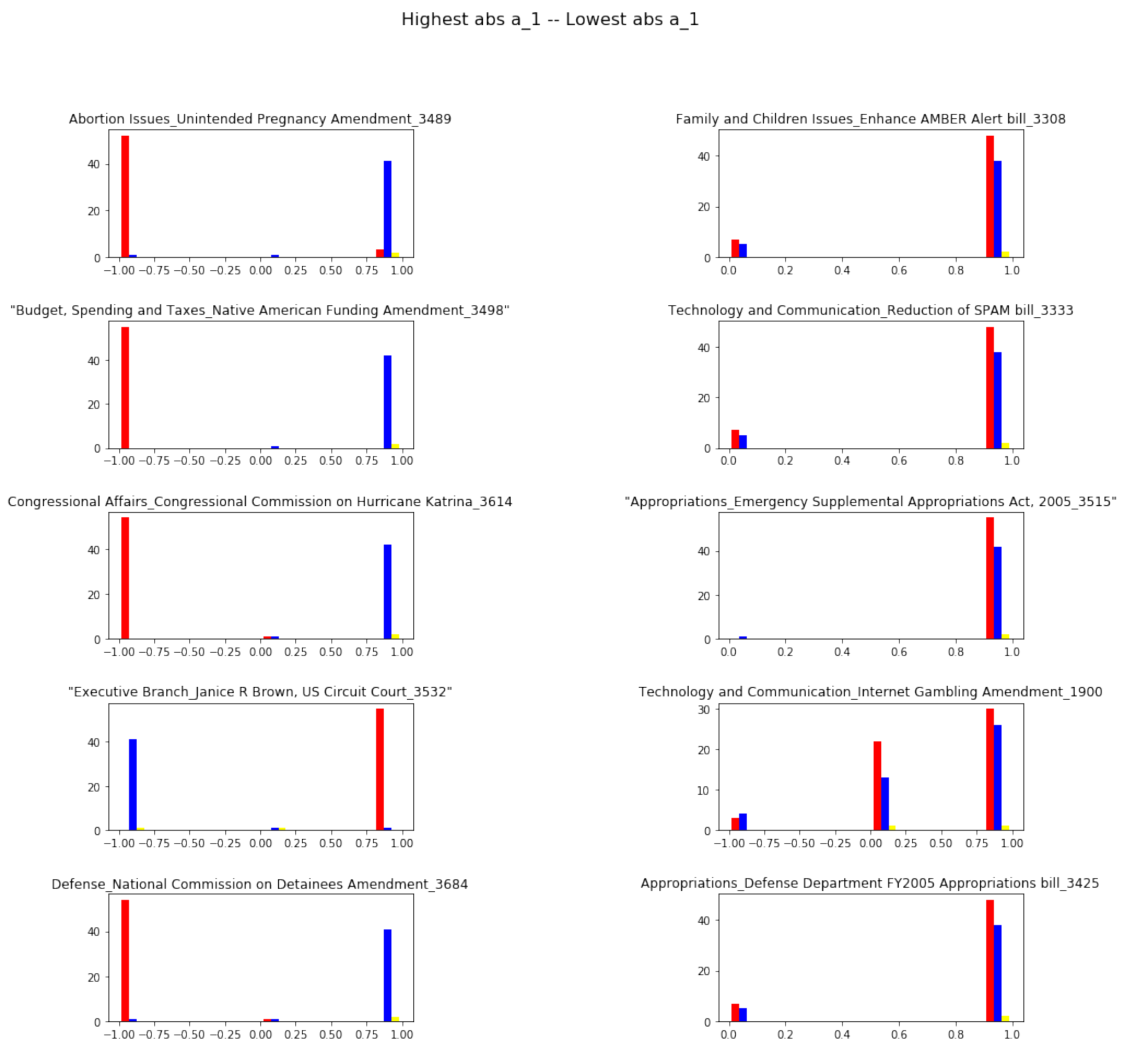
0.2.3 #TODO Part d ii) Comment on how the voting looks like for bills with most variance and bills with least variance

For the bills with the least variance, it appears that very little of the variance can be explained to by the bills where everyone voted for the same thing. Meanwhile the bills with the least variance where those where voting was almost purely along party lines.

0.2.4 Approach 2: We find the projection of the basis vector corresponding to each bill on to the first principal components and choose those bills with highest absolute value of projections. Note that this is equivalent to choosing bills based on highest absolute values of a_1.

In [21]:
```
# Recall that a_1_scores holds the projection onto the first principal component
a_1_flat = np.ndarray.flatten(a_1) # first, flatten the a_1 of len 542
abs_a_1 = np.abs(a_1_flat)

sorted_idxes = np.argsort(-abs_a_1) #in decreasing order
print(sorted_idxes.shape)

top_5_a1 = [bills[sorted_idxes[i]] for i in range(5)]
bot_5_a1 = [bills[sorted_idxes[-1-i]] for i in range(5)]


fig, axes = plt.subplots(5,2, figsize=(15,15)) # 1 plot to make things easier to see

for i in range(5):
    idx = sorted_idxes[i]

    X_red_c = X_original[np.array(affiliations) == 'Red',idx]
    X_blue_c = X_original[np.array(affiliations) == 'Blue',idx]
    X_yellow_c = X_original[np.array(affiliations) == 'Yellow',idx]

    axes[i,0].hist([X_red_c, X_blue_c, X_yellow_c], color = ['red', 'blue', 'yellow']
    axes[i,0].set_title(bills[idx])


for i in range(1,6):
    idx2 = sorted_idxes[-i]

    X_red_c2 = X_original[np.array(affiliations) == 'Red',idx2]
    X_blue_c2 = X_original[np.array(affiliations) == 'Blue',idx2]
    X_yellow_c2 = X_original[np.array(affiliations) == 'Yellow',idx2]

    axes[i-1,1].hist([X_red_c2, X_blue_c2, X_yellow_c2], color = ['red', 'blue', 'yel
    axes[i-1,1].set_title(bills[idx2])

plt.subplots_adjust(hspace=0.5, wspace = 1)
fig.suptitle('Highest abs a_1 -- Lowest abs a_1', fontsize=16)
plt.show()
```

```
(542,)
```



0.2.5 #TODO Part d iii) Comment on how the voting looks like for bills with highest and lowest absolute values of a_1.

Next let us compare the bills found by the two approaches However, it appears to be the case that those bills with the largest a_1 values are those with the largest polarity between parties, while for those with the smallest a_1 values, it appears that the opposite holds true.

```
In [22]: # The bills that are the same in both the top and bottom 10 using these different met
         print('Number of common bills in top:', len(np.intersect1d(top_5 ,top_5_a1)))
         print('Number of common bills in bottom :', len(np.intersect1d(bot_5 ,bot_5_a1)))
```

```
Number of common bills in top: 0
Number of common bills in bottom : 0
```

0.2.6 #TODO Part d iv) Are the bills in the two approaches the same? What do you think is the reason for the difference?

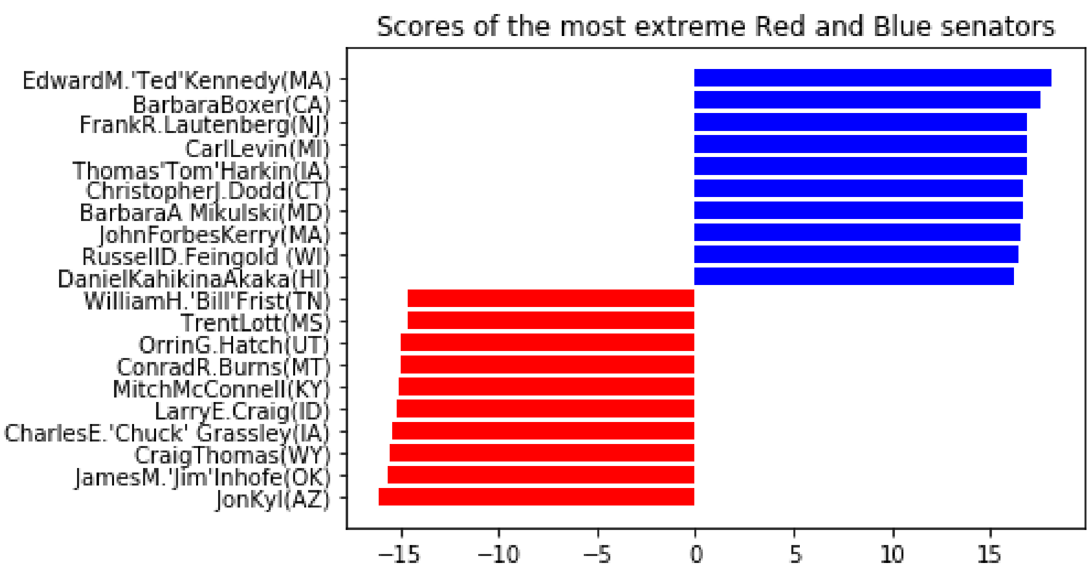
The bills in the two approaches are not the same. This is due to how in both problems we are finding the same quantity, but in one formulation we take projections along the directtion of maximal variance, the first principal component, while in the other one you take projections along e1.

0.2.7 Part e) Finally, we will look at the scores for senators along the first principal direction and make the following classifications for senators:

a) Senators with the top 10 most positive scores and top 10 most negative scores are classified as most "extreme".
b) Senators with the 20 scores closest to 0 are classified as least "extreme".

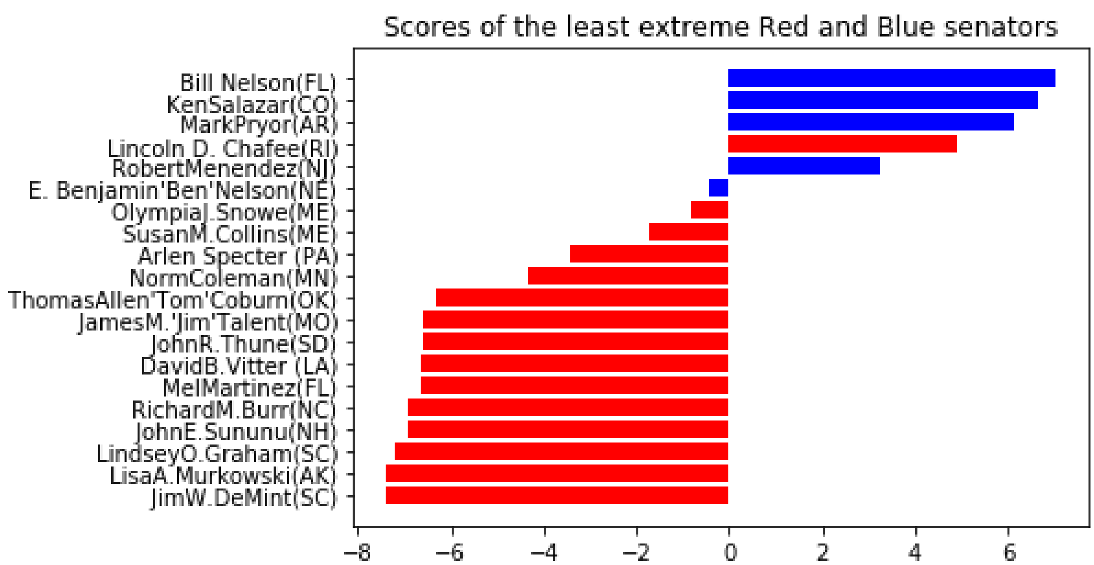
Most extreme senators

In [24]:
```
senators = senator_df.columns.values[3:]

senator_scores = f(X,a_1)
complete_sort_indices = np.argsort(senator_scores)

sort_indices = np.hstack([complete_sort_indices[:10], complete_sort_indices[-11:-1]])
senators_sorted = senators[sort_indices]
senator_scores_sorted = senator_scores[sort_indices]
affiliations = np.array(affiliations)
affiliations_sorted = affiliations[sort_indices]

plt.barh(y = senators_sorted, width = senator_scores_sorted, color = affiliations_sor
plt.title('Scores of the most extreme Red and Blue senators')
plt.show()
```



Least extreme senators

```
In [25]: senator_scores = f(X,a_1)

         # print(np.sort(np.abs(senator_scores)))
         complete_sort_indices = np.argsort(np.abs(senator_scores))[:20]

         senator_scores_le= senator_scores[complete_sort_indices]
         senators_le = senators[complete_sort_indices]
         affiliations = np.array(affiliations)
         affiliations_le = affiliations[complete_sort_indices]
         sort_indices = np.argsort(senator_scores_le)
         senators_sorted = senators_le[sort_indices]
         senator_scores_sorted = senator_scores_le[sort_indices]
         affiliations_sorted = affiliations_le[sort_indices]

         plt.barh(y = senators_sorted, width = senator_scores_sorted, color = affiliations_sor
         plt.title('Scores of the least extreme Red and Blue senators')
         plt.show()
```



0.2.8 #TODO Comment on the sign of scores vs party affiliations.

It appears to be the case that amongst both least extreme senators, it appears to be the case that republican voters tend to have more negative scores, which translates to voting no on bills. In terms of the most extreme senators, it appears that an equal portion of voters are extreme by these metrics.

Homework 3 EE 127

oscar.g.ortega.5

February 2019

1 Interpreting the Data Matrix

a:

feature means = numpy.mean(X,axis = 0) and the length of the vector will be of size m because there are m features.

b:

feature stddevs = numpy.std(X, axis = 0) will be the correct command and the length of the resultant vector will be of size n

c:

If we want every feature "centered", for every data point, we need to subtract the average of each corresponding feature to make the features 0 mean

d:

If we want every feature "standardized" we would first "center" the data points and then proceed to divide each feature by the square root of the std deviation of that feature

e:

The covariance matrix is of size $m \times m$ as we are dealing with the number of features

f:

Assuming all the features are 0 mean, we could find the covariance of all the features by dotting all the features with eachother and seeing how they are related. In other words, if I want to find the covariance between feature x and feature y, I would dot rows x and y with eachother and then divide that quantity by the number of points, n. Because we want to do this for every combination of features, this is equal to the formula below

$$= \frac{1}{n}\sum_{i=1}^{n}(x^i)^T x^i$$

$$= \frac{X^T X}{n}$$

g:

$$c_{i,j} = \frac{1}{n}(x^i)^T x^j$$

h:

$$u \in \mathbb{R}$$

i: Recall

$$proj_a(b) = a^T b$$

$$\rightarrow proj_u(x^{(i)}) = x^{(i)\,T} a$$

Furthermore, recall

$$X := \begin{vmatrix} x_1^T \\ x_2^T \\ . \\ . \\ . \\ {x_n}^T \end{vmatrix}$$

$$Xu = \begin{vmatrix} proj_u(x_1) \\ proj_u(x_2) \\ .. \\ . \\ proj_u(x_n) \end{vmatrix}$$

which means we can define $z = Xu$

j:

$$z = Xu$$

$$\frac{1}{n}u^T X^T Xu = \frac{1}{n}z^T z = \frac{1}{n}\langle z, z\rangle = \frac{1}{n}\sum_i z_i^2 = \frac{1}{n}\sum_i proj_u(z)_i^2 = Var(z)$$

2 PCA and low-rank compression

(a)

$$v* = argmin_v \|x_0 + vu - x\|_2$$
$$= argmin_v \|x_0 + vu - x\|_2^2$$
$$= \langle x_0 + vu - x, x_0 + vu - x\rangle$$
$$= 2\langle x_0, vu\rangle - 2\langle x, vu\rangle + \langle uv, uv\rangle$$

$$= argmin_v 2(-x + x_0)^T vu + v^T v$$

$$\frac{\partial f}{\partial v} = 2(-x + x_0)^T u + 2v$$

$$\rightarrow v = (x - x_0)^T u$$

This follows from the pythagorean identity: $a^2 + b^2 = c^2$ where c^2 is your hypotenuse and a^2 and b^2 are the other two angles of your right triangle.

$$\|x - x_0\|_2^2 = d^2 + ((x - x_0)^T u)^2$$
$$d^2 = \|x - x_0\|_2^2 - ((x - x_0)^T u)^2$$

(b)

$$\rightarrow argmin_v \|x_0 + vu - x\|_2^2 = argmax_v \|x - x_0\|_2^2 - \|x_0 + vu - x\|_2^2$$

Because our data is centered, let $x_0 = 0$

$$\rightarrow argmin_v \|vu - x\|_2 = argmax_v (\|x\|_2^2 - \|vu - x\|_2^2)$$
$$min_v \|vu - x\|_2^2 = ((x)^T u)^2)$$

$$argmin_{u\in\mathbf{R}^m:\langle u,u\rangle=1} \sum_{i=1}^{n} min_{v_i\in\mathbf{R}} \|x_i - v_i u\|_2^2$$

$$argmin_{u\in\mathbf{R}^m:\langle u,u\rangle=1} \sum_{i=1}^{n} max_{v_i\in\mathbf{R}} \|x_i\|^2 - \|x_i - v_i u\|_2^2$$

$$argmin_{u\in\mathbf{R}^m:\langle u,u\rangle=1} \sum_{i=1}^{n} \|x_i\|^2 - ((x_i^T u)^2)$$

$$argmax_{u\in\mathbf{R}^m:\langle u,u\rangle=1} \sum_{i=1}^{n} ((x_i^T u)^2)$$

$$argmax_{u\in\mathbf{R}^m:\langle u,u\rangle=1} \sum_{i=1}^{n} u^T x_i x_i^T u$$

$$argmax_{u\in\mathbf{R}^m:\langle u,u\rangle=1} u^T \frac{\sum_{i=1}^{n}(x_i x_i^T)}{n} u$$

$$argmax_{u \in \mathbf{R}^m : \langle u,u \rangle = 1} u^T C u$$

(c) By the singular value decomposition theorom: $\forall A \in R^{m,n}$ A can be represented as

$$A = \sum_{i=1}^{n} \sigma_i u_i v_i^T$$

where $\sigma_1, \sigma_2, ..., \sigma_n$ are the singular values of a. The $\{u_i\}_i \in \mathbb{R}^n$ and the $\{v_i\}_i \in R^m$

$$Rank(A) = 1 \rightarrow A = \sigma_1 u_1 v_1^T$$

let $\sigma_1 v_1 = v'$

$$\rightarrow A = u v'^T : u \in \mathbf{R^n} and v \in \mathbf{R^m}$$

(d) Consider the following minimization:

$$argmin_{Y:rank(Y)=1} \|X - Y\|_F$$

$$= argmin_{Y:rank(Y)=1} \|X - Y\|_F^2$$

From part c we know that if it is the case the rank of Y is one, then $Y = uv^T$ for some $v \in \mathbb{R}^n$ and $u \in \mathbb{R}^m$ where u can arbitrarily be scaled to a unit norm.

$$= argmin_{v \in \mathbb{R}^m : \langle v,v \rangle = 1} \sum_{i=1}^{n} \sum_{j=1}^{m} \min_{u_i \in \mathbb{R}} |X_{i,j} - (uv)_{i,j}|^2$$

$$= argmin_{v \in \mathbb{R}^m : \langle v,v \rangle = 1} \sum_{i=1}^{n} \min_{u_i \in \mathbb{R}} \|X_i - u_i v\|_2^2$$

3 SVD Transformation

(a) $Let v_1, v_2, ..., v_n$ be the columns of v, by construction, we know that the columns of v form an orthonormal basis for v in $\mathbf{R}^n$

$$\forall x \in R^n : V^T x = \sum_{i=1}^{n} \langle v_i, x \rangle x_i$$

Furthermore

$$\forall x, y \in R^n : V^T x = V^T y \rightarrow \sum_{i=1}^{n} \langle v_i, x \rangle x_i = \sum_{i=1}^{n} \langle v_i, y \rangle y$$

$$\rightarrow V^T (x - y) = V^T (x) - v^T (y) = 0 \rightarrow x = y$$

Therefore the representation is unique. Rest of the problem on the python notebook.

4 Senator Problem

On the python notebook.

5 Matrix Norms

(a)

$$v := argmax_{x\neq 0}\frac{\|Ax\|_p}{\|x\|_p}$$

$$u := argmax_{\|x\|=1}\|Ax\|_p$$

$$max_{x\neq 0}\frac{\|Ax\|_p}{\|x\|_p} \geq max_{\|x\|=1}\|Ax\|_p$$

proof:

$$v := argmax_{x\neq 0}\frac{\|Ax\|_p}{\|x\|_p} \to \forall x\frac{\|Av\|_p}{\|v\|_p} \geq \frac{\|Ax\|_p}{\|x\|_p}$$

$$\to \frac{\|A_v\|p}{\|v\|_p} \geq \frac{\|A_u\|_p}{\|u\|_p}$$

$$max_{x\neq 0}\frac{\|Ax\|_p}{\|x\|_p} max_{\|x\|=1}\|Ax\|_p$$

Proof:

$$u := argmax_{\|x\|=1}\|Ax\|_p \to \forall x\|Au\|_p \geq \|Ax\|_p$$

$$\to \|Au\|_p \geq \|A\frac{v}{\|v\|_p}\|_p$$

$$\frac{\|A_u\|_p}{\|u\|_p} \geq \frac{\|Av\|_p}{\|v\|_p}$$

Therefore, the two definitions are equivalent.

(b)

(i)Let $v \in \mathbb{R}^n : \|v\|_1 = 1$ and consider $A = \begin{vmatrix} A_1 & A_2 & \dots & A_n \end{vmatrix}$ s.t

$$\|A_1\|_1 \geq \|A_2\|_1 \geq \dots \geq \|A_n\|_1$$

$$\|Av\|_1 = \sum_{i=1}^{n} v_i\|A_i\|_1$$

$$\to max_{\|x\|_1=1}\|Av\|_1 = \|A_1\|_1$$

(ii) $Let v \in \mathbb{R}^n : \|v\|_\infty = 1$ and consider $A = \begin{vmatrix} A_1 & A_2 & \dots & A_m \end{vmatrix}^T$ s.t

$$\|A_1\|_\infty \geq \|A_2\|_\infty \geq \dots \geq \|A_m\|_\infty$$

$$\rightarrow \|Ax\|_\infty = max_j(\sum_{i=1}^{n} x_i A_{i,j})_{j=1,2,...,m}$$

$$\rightarrow_{x:\|x\|_\infty=1} \|Ax\|_\infty = \begin{vmatrix} 1_1 & 1_2 & \dots & 1_n \end{vmatrix}^T$$

$$\rightarrow = max_x \|Ax\|_\infty = max_j(\|A_j\|_\infty)_{j=1,2,...,m}$$

(iii)

$$max_x \|Ax\|_2 = max_x \langle Ax, Ax \rangle$$
$$= max_x(x^T A^T Ax)$$

$A^T A$ symmetric $\rightarrow U \in R^{n,n}$ orthonormal and Λ diagonal s.t $A = U \Lambda U^T$

$$max_x(x^T U \Lambda U^T x)$$

Let $x'^T = x^T U$ and $x' = U^T x$ $\|x'\| = \|x\|$ which shows $= max_x(x^T A^T Ax) == max'_x(x'^T \Lambda x')$ which is maximized when all the weight of x is focused on the first component.

$$\rightarrow = \sqrt{\lambda_{max}(A^T A)}$$
$$= \sigma_x(A)$$

the largest singular value of A.

(c)

$$\|Av\|_p \geq \|A\|_p \|v\|_p$$

Proof:

$$\forall v \in V : \frac{\|Av\|_p}{\|v\|_p} \leq max_x \frac{\|Ax\|_p}{\|x\|_p}$$
$$\frac{\|Av\|_p}{\|v\|_p} \leq \|A\|_p$$
$$\rightarrow \|Av\|_p \leq \|A\|_p \|v\|_p$$

let $v \in V$

$$\|ABv\|_p \geq \|A\|_p \|Bv\|_p \leq \|A\|_p \|B\|_p \|v\|_p$$
$$\rightarrow \forall v \in V : \frac{\|ABv\|_p}{\|v\|_p} \leq \|A\|_p \|B\|_p$$
$$\|AB\|_p \leq \|A\|_p \|B\|_p$$

(d) Based on the inequalities above:

$$\frac{1}{\sqrt{nm}} \|A\|_1 \|A\|_\infty \leq \|A\|_2^2 \leq \sqrt{mn} \|A\|_1 \|A\|_\infty$$

$$\frac{1}{\sqrt{nm}}\|A\|_1 \leq \frac{\|A\|_2^2}{\|A\|_\infty} \leq \sqrt{mn}\|A\|_1$$

$$\frac{\|A\|_1}{\sqrt{nm}\|A\|_2^2} \leq \frac{1}{\|A\|_\infty} \leq \frac{\sqrt{mn}\|A\|_1}{\|A\|_2^2}$$

$$\frac{\sqrt{nm}\|A\|_2^2}{\|A\|_1} \geq \|A\|_\infty \geq \frac{\|A\|_2^2}{\sqrt{mn}\|A\|_1}$$

$$\frac{\sqrt{nm}n\|A\|_1^2}{\|A\|_1} \geq \|A\|_\infty \geq \frac{\|A\|_1^2}{m\sqrt{nm}\|A\|_1}$$

$$n\sqrt{nm}\|A\|_1 \geq \|A\|_\infty \geq \frac{\|A\|_1}{m\sqrt{nm}}$$

(e) From part a: We know A symmetric implies $tr(A) = \sum_i \lambda i$ by the spectral theorom where λ_i is the ith eigenvalue of A:

$$\|A\|_2 = \sqrt{\lambda_{max}(A^T A)} \leq \sqrt{\sum_{i=1}^{n} \lambda_i (A^T A)}$$

$$\|A\|_2 \leq \|A\|_F$$

Let $\lambda_1, \lambda_2, ..., \lambda_r$ be the first eigenvalues of $A^T A$ where $\lambda_1 \geq \lambda_2, ...\lambda_r \geq 0$ and let the remaining $n - r$ eigenvalues be 0.

$$\sqrt{\sum_{i=1}^{n} \lambda_i (A^T A)} = \sqrt{\sum_{i=1}^{r} \lambda_i (A^T A)} \leq \sqrt{\sum_{i=1}^{r} \lambda_{max} (A^T A)}$$

$$\|A\|_F \leq \sqrt{r}\sqrt{\lambda_{max} A^T A}$$

$$\|A\|_F \leq \sqrt{r}\|A\|_2$$

6 Connected Graphs and Laplacians

(a) The laplacian of the following undirected graph is as follows:

$$\begin{vmatrix} 2 & -1 & 0 & 0 & -1 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & -1 & 0 & -1 & 3 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \end{vmatrix}$$

(b) to show symmetry holds for generic laplacian graphs: we must show $A_{i,j} = A_{j,i} \forall i, j \in V \times V$

By definition of the Laplacian matrix: when $i = j$ $d(i) = A_{(i,j)} = A_{(j,i)}$

Furthermore, because we are dealing with undirected graphs,

$$(i, j) \in V \times V \rightarrow (j, i) \in V \times V$$

$$\rightarrow \forall (i, j) \in V \times V : A(i, j) = A(j, i) = -1 : i \neq j$$

Therefore, the laplacian matrix is symettric.
(c) The laplacian matrix is positive semidefinite:
proof:
(Credit to Wikipedia on the the following decomposition of the Laplacian Matrix)
Let D = the degree matrix of the graph: i.e d(i,i) = deg(i) $\forall i \in V$
Let A = the adjacency matrix of the graph: i.e$\forall (i, j) \in V \times V : A_{i,j} = 1$
Then we can view $L = D - A$
Let $x \in R^n$

$$\rightarrow x^T Lx = x^T (D - A)x = x^T Dx - x^T Ax$$

$$= \sum_{i=1}^{n} deg(i)x_i^2 - \sum_{(i,j)\in E} 2x_i x_j$$

$$= \sum_{i=1}^{n} \sum_{(i,j)\in E} x_i^2 - \sum_{(i,j)\in E} 2x_i x_j$$

$$= \sum_{(i,j)\in E} 2x_i^2 - \sum_{i,j\in E} 2x_i x_j$$

'sum of vertex degrees is equal to twice the number of edges'

$$= \sum_{(i,j)\in E} x_i^2 + x_j^2 - x_i x_j$$

$$= \sum_{(i,j)\in E} (x_i - x_j)^2$$

note how we are overcounting by a factor of two:

$$= \frac{1}{2} \sum_{(i,j) \in E} (x_i - x_j)^2$$

(d) Consider the vector $u = \lambda_i \begin{vmatrix} 1 & 1 & \dots & 1 \end{vmatrix}^T \in \mathbb{R}^n$.

$$\rightarrow Lu = (D - A)u$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} deg(i) - \mathbf{1}((i, j \in E))$$

$$\sum_{i=1}^{n} deg(i) - deg(i) = 0$$

$\rightarrow$ 0 is an eigenvalue of L with the demonstrated eigenvector

(e) Proof:

Consider a disjoint graph with two connected components $G = (V, E)$ and $G' = (V', E')$ where the cardinalities of V and V' are n and n' respectively.

$$\rightarrow L_{G \bigcup G'} = \begin{vmatrix} L_G & 0 \\ 0 & L_{G'} \end{vmatrix}$$

Where L_G and $L_{G'}$ are the laplacians of G and G'.

Let $u = \begin{vmatrix} \lambda 1_1 & \lambda 1_2 & \dots & \lambda 1_n & 0_{n+1} & 0_{n+2} & \dots & 0_{n+n'} \end{vmatrix}^T = \begin{vmatrix} \lambda 1 | 0 \end{vmatrix} \in \mathbb{R}^{n+n'}$

$$\rightarrow L_{G \bigcup G'} u = \begin{vmatrix} L_G \lambda 1 & 0 \\ 0 & L_{G'} 0 \end{vmatrix} = 0$$

This can be seen from part d.

$\rightarrow null(L)$ is not simple.