ECE 57000 Assignment 2 Exercises

Name: Mohmmad Alwakeel

Exercise 1 (10/100 points)

In this exercise, you will need to write a simple function that reverses and doubles the values in a list. For example: input `[1,2,3]`, output `[6,4,2]`.

```
In [5]: def reverse_double(input:list)->list:

          # <YOUR CODE>
            return [ele*2 for ele in reversed(input)]

        A = [2,5,3,9,7]
        print(reverse_double(A))
```

```
[14, 18, 6, 10, 4]
```

Exercise 2 (20/100 points)

In this exercise, you will need to help visualize several different distrubtions.

Task 1

1. Using numpy to generate a vector **D** with the following property:

   - Each element is in a normal distribution.
   - Vector has the shape **2000x1**
2. Reshape the vector **D** into **1000x2**
3. Plot the graph in the following way:

   - Create a figure of size 6 by 6
   - Treat the two columns of the array **D** as the **x** and **y** coordinates of 2D points. Use scatter() (https://matplotlib.org/3.2.2/api/_as_gen/matplotlib.pyplot.scatter.html) to visualize all the spots and set the marker size to be 1
   - Let the plot shows the range `[-5,5]x[-5,5]`
   - Give the plot a title (indicating the shape of **D**), and also label the x-axis and y-axis

Note: It is always important to include necessary information (e.g. label, legend, title) so that readers won't get confused.

```
In [448]:  import numpy as np
           import matplotlib.pyplot as plt

           #  <YOUR CODE>
           np.random.seed(1)
           D = np.random.normal(0,1,2000)
           #D = np.random.randint(low=0, high = 5, size=2000)
           D = D.reshape(-1,2)
           Y = D[:,1]
           X = D[:,0]
           plt.figure(figsize=(6,6))

           plt.scatter(X,Y,s=1)
           plt.xlim(-5,5)
           plt.ylim(-5,5)
           plt.title('Normal Distribution Data Visualization D (std= 1)')
           plt.xlabel('x-axis D[col = 0]')
           plt.ylabel('y-axis D[col = 1]')
           plt.show()
```
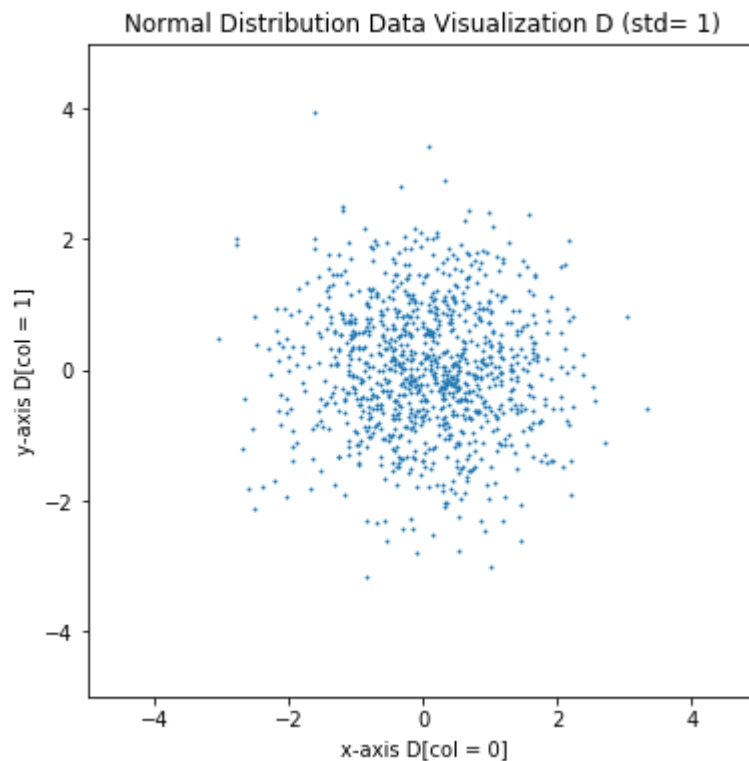

Task 2

1. Create an array **R** =
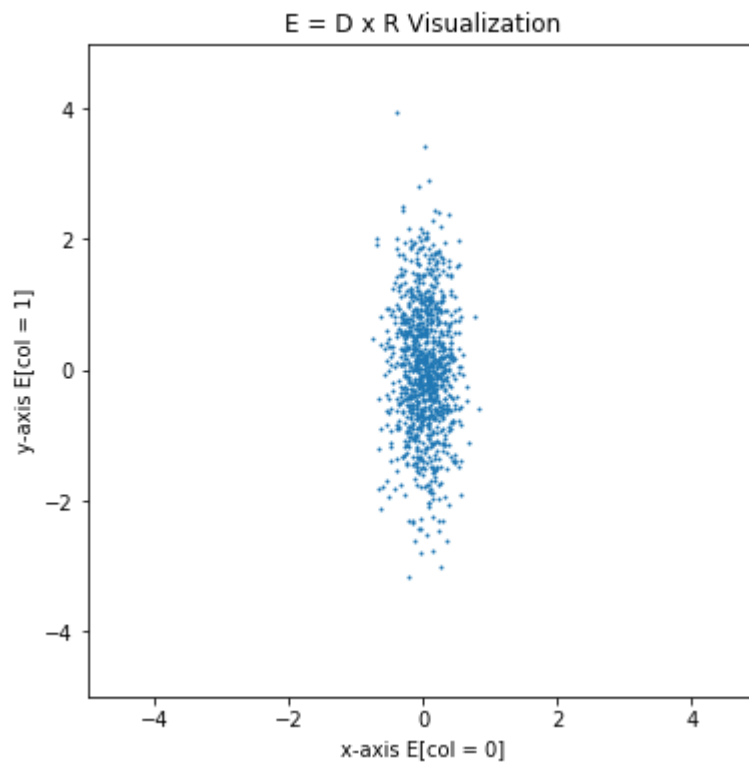
$$\begin{bmatrix} 0.25 & 0 \\ 0 & 1 \end{bmatrix}$$

2. Compute **E = D × R**.
3. Repeat Step 3 above for **E**. (Title: shape of **E**)

In [449]:
```python
#  <YOUR CODE>
R = np.array([[0.25, 0], [0, 1]])

E = np.matmul(D,R)

Y = E[:,1]
X = E[:,0]
plt.figure(figsize=(6,6))

plt.scatter(X,Y,s=1)
plt.xlim(-5,5)
plt.ylim(-5,5)
plt.title('E = D x R Visualization')
plt.xlabel('x-axis E[col = 0]')
plt.ylabel('y-axis E[col = 1]')
plt.show()
```


Task 3

1. Create an array **R** =

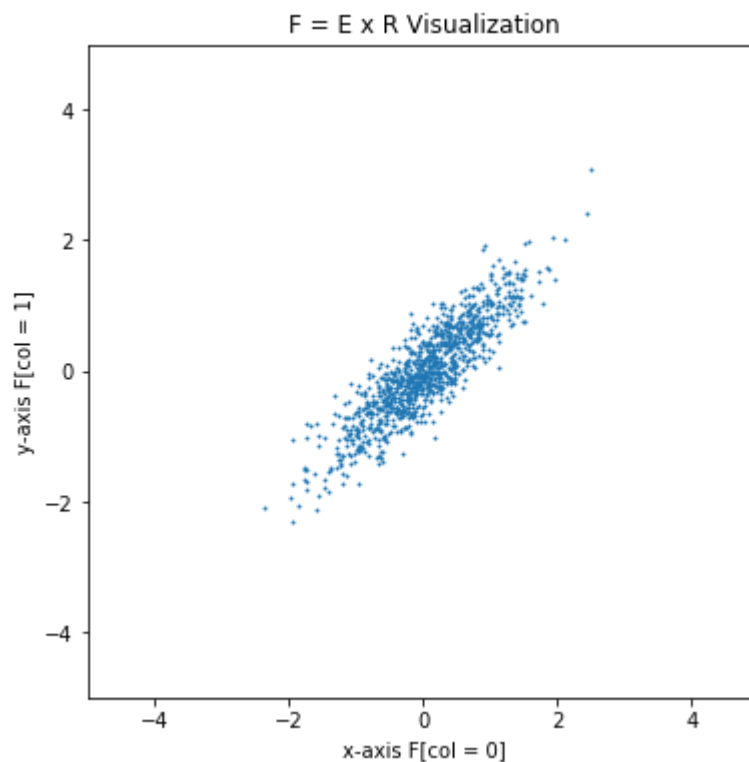$$\begin{bmatrix} \sqrt{(2)}/2 & -\sqrt{(2)}/2 \\ \sqrt{(2)}/2 & \sqrt{(2)}/2 \end{bmatrix}$$

2. Compute **F** = **E** $\times$ **R**.
3. Repeat Step 3 above for **F**. (Title: shape of **F**)

```
In [450]: #  <YOUR CODE>
          R = [[(2**(1/2))/2, -(2**(1/2))/2], [(2**(1/2))/2, (2**(1/2))/2]]

          F = E @ R

          Y = F[:,1]
          X = F[:,0]
          plt.figure(figsize=(6,6))
          plt.scatter(X,Y,s=1)
          plt.xlim(-5,5)
          plt.ylim(-5,5)
          plt.title('F = E x R Visualization')
          plt.xlabel('x-axis F[col = 0]')
          plt.ylabel('y-axis F[col = 1]')
          plt.show()
```


Exercise 3 (70/100 points)

Task 1: Generate a sparse matrix

1. Generate a matrix **X** with size 100x50 with each element randomly picked from a uniform distribution **U**[0,1].
2. Use logical(boolean) indexing to set the elements in **X** to **0** whenever the value of the element is smaller than 0.85 (In this way, you should get the matrix to have roughly 85% of its elements zero's).
3. Use the function csr_matrix() (https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csr_matrix.html#scipy.sparse.csr_matrix) to convert the matrix **X** into sparse matrix and call it **X_sparse**.

```
In [355]: import numpy as np
          from scipy.sparse import csr_matrix

          #  <YOUR CODE>

          np.random.seed(1)
          X = np.random.uniform(0,1.0, size=(100,50))
          X[X < 0.85] = 0
          X_sparse = csr_matrix(X)

          print(f'X has type {type(X)} and has {100-np.sum(X!=0)/50}% of zeros')
          print(f'X_sparse has type {type(X_sparse)} and has {100-np.sum(X_sparse!=0)/5
          0}% of zeros')
```

```
X has type <class 'numpy.ndarray'> and has 84.86% of zeros
X_sparse has type <class 'scipy.sparse.csr.csr_matrix'> and has 84.86% of zer
os
```

Task 2: Construct the power iteration function

Following the algorithm in the instructions notebook, write a function that takes a sparse matrix **X** and number of iterations as input and returns the top right singular vector of the centered matrix as output. We have provided some starter code and you need to fill in the rest.

First Method: ~ Matrix dot product

```
In [422]: #right
          def power_iter(X, num_iter:int):
              v = np.random.randn(X.shape[1])  # Initialize with random vector with shap
          e (d,)
              one_vec = np.ones(X.shape[0])  # All ones vector with shape (n,)
              mu_row_matrix = np.mean(X, axis=0)  # Returns a 1 row matrix with shape
           (d, 1) since X is of "matrix" type
              mu = np.array(mu_row_matrix).squeeze()

              for _ in range(num_iter):

                  v = X.T.dot(X.dot(v)) + mu.dot((one_vec.T).dot(one_vec)).dot((mu.T).do
          t(v)) - mu.dot((one_vec.T).dot(X.dot(v))) - ((X.T).dot(one_vec)).dot((mu.T).do
          t(v))
                      #- mu.dot((one_vec.T).dot(X.dot(v))) - ((X.T).dot(one_vec)).dot((mu.
          T).dot(v))
                  v = v/np.linalg.norm(v)
              return v

          v1_yours = power_iter(X_sparse,1000).squeeze()
          print(v1_yours.shape)
```

```
(50,)
```

Second Method: ~ Matrix multiplication

```
In [433]:  def power_iter(X, num_iter:int):
               v = np.random.randn(X.shape[1])  # Initialize with random vector with shap
           e (d,)
               one_vec = np.ones(X.shape[0])  # All ones vector with shape (n,)
               mu_row_matrix = np.mean(X, axis=0)  # Returns a 1 row matrix with shape
            (d, 1) since X is of "matrix" type
               mu = np.array(mu_row_matrix).squeeze() # Convert from a sparse column matr
           ix to a dense 1D array with shape (d,)

               v = v.reshape(len(v),1)
               one_vec = one_vec.reshape(len(one_vec),1)
               mu = mu.reshape(len(mu),1)

               #   <YOUR CODE>

               for _ in range(num_iter):
                   v = (X-(one_vec@mu.T)).T @ (X-(one_vec@mu.T)) @ v
                   v = v/np.linalg.norm(v)


               S = np.array(v).squeeze()
               return S
               #return V[num_iter-1]
           v1_yours = power_iter(X_sparse,1000).squeeze()
           print(v1_yours.shape)
```

```
(50,)
```

Task 3: Verifying your top singular vector

Using any method you like to verify the vector that is computed by your function is indeed the top right singular vector of the **centered** data matrix. First write another function that outputs the top right singular vector for sure (you can use the function [svd() (https://numpy.org/doc/stable/reference/generated/numpy.linalg.svd.html)](https://numpy.org/doc/stable/reference/generated/numpy.linalg.svd.html), note that it returns V^T instead of V). Then, the provided code will compute the mean absolute error (MAE) between the two functions you wrote. (Note: The provided evaluation code will correct for the fact that the two vectors can be the negative of each other singular value decomposition is only unique up to signs). The MAE should be close to machine precision (i.e., it should be less than about `1e-15`).

Note: This is for testing the correctness of your algorithm. It is often a very good idea to write simple checks of your code as you write it to avoid bugs early on in your development process. Do not worry about efficiency for this exercise.

First Method: ~ Matrix subtraction

In [441]:
```python
def verify_v1(X):
    # Compute the top right singular vector using other methods
    #  <YOUR CODE>

    one_vec = np.ones(X.shape[0])
    mu_row_matrix = np.mean(X, axis=0)
    mu = np.array(mu_row_matrix).squeeze()

    u, s, vh = np.linalg.svd(X - mu)
    #print("u = ",u.shape,"s =", s.shape, "vh",vh.T.shape)

    return vh[0]

# Note here we just pass in the dense 2D array `X`
#  which represents the same matrix as `X_sparse`
v1_simple = verify_v1(X).squeeze()
# Compute a sign corrected difference between the vectors
#  (accounting for the fact that SVD is only unique up to signs)
diff_sign_corrected = np.sign(v1_yours[0]) * v1_yours - np.sign(v1_simple[0])
* v1_simple
mae_corrected = np.mean(np.abs(diff_sign_corrected))
print(f'The average absolute difference of the two function output is {mae_cor
rected}')
```

The average absolute difference of the two function output is 1.1559286514084
554e-15

Second Method: ~ Matrix multiplication

```
In [440]:  def verify_v1(X):
             # Compute the top right singular vector using other methods
             #   <YOUR CODE>

               one_vec = np.ones(X.shape[0])  # All ones vector with shape (n,)
               mu_row_matrix = np.mean(X, axis=0)  # Returns a 1 row matrix with shape
           (d, 1) since X is of "matrix" type
               mu = np.array(mu_row_matrix).squeeze()

               one_vec = one_vec.reshape(len(one_vec),1)
               mu = mu.reshape(len(mu),1)

               u, s, vh = np.linalg.svd(X - one_vec @ mu.T)
               #print("u = ",u.shape,"s =", s.shape, "vh",vh.T.shape)

               return vh[0]

           # Note here we just pass in the dense 2D array `X`
           #   which represents the same matrix as `X_sparse`
           v1_simple = verify_v1(X).squeeze()
           # Compute a sign corrected difference between the vectors
           #   (accounting for the fact that SVD is only unique up to signs)
           diff_sign_corrected = np.sign(v1_yours[0]) * v1_yours - np.sign(v1_simple[0])
           * v1_simple
           mae_corrected = np.mean(np.abs(diff_sign_corrected))
           print(f'The average absolute difference of the two function output is {mae_cor
           rected}')
```

The average absolute difference of the two function output is 1.1559286514084
554e-15

(Optional and ungraded, 0 points) Task 4: Going beyond

- In what scenarios we might find the power iteration method useful?
  - Google's original ranking algorithm called "PageRank" uses a variant of this power iteration on very sparse graphs that represent connections between websites. See PageRank (https://en.wikipedia.org/wiki/PageRank).
- Can you optimize your algorithm further by avoiding reusing computations?
- Below, try your method with a much larger sparse X matrix (maybe 1000 x 1000 with 0.01% nonzeros, i.e. very sparse) and time the difference. For very short code snippets, you can use the `timeit` Python package or just use `time.time()` to capture the start and end times (subtracting them gets you the time in seconds).

```
In [ ]:
```