Ex .1 - Numerical stability

1. A higher precision in a big data context will grant a method greater stability, the ability to react more precisely with small perturbation.
2.

```
tic
for a = 1:100
    x = rand(1000,100);
    svd(x);
end
toc

tic
for a = 1:100
    x = rand(1000,100);
    svd(x');
end
toc

tic
for a = 1:100
    x = rand(1000,100);
    A = eig(x * x');
end
toc

tic
for a = 1:100
    x = rand(1000,100);
    A = eig(x' * x);
end
toc
```

The result is as follows:

```
Elapsed time is 0.364459 seconds.
Elapsed time is 0.299080 seconds.
Elapsed time is 5.254796 seconds.
Elapsed time is 0.193208 seconds.
```

We can see that calculate the eigenvalue of XX^T takes significantly longer time than other methods.

3. We first import x.

```
x = [-9 11 -21 63 -252;

     70 -69 141 -421 1684;

     -575 575 -1149 3451 -13801;

     3891 -3891 7782 -23345 93365;

     1024 -1024 2048 -6144 24572; ];
```

a) Then we run the test for the eigenvalue. We generate 1000 δX values and calculate average. Then we use each δX to calculate the eigenvalue of X + δX, and we subtract that value from the eigenvalue of X and calculate the average error. Finally we compare these two errors.

```
err_eig = [0;0;0;0;0;];
p_total_a = [0 0 0 0 0;
            0 0 0 0 0;
            0 0 0 0 0;
            0 0 0 0 0;
            0 0 0 0 0;];

for a = 1:1000
    pertur_a = eps(x);
    p_total_a = p_total_a + pertur_a;
    err_eig = err_eig + eig(x + pertur_a) - eig(x);
end
err_eig = err_eig / 1000
p_total_a = p_total_a / 1000
```

The result is follows:

```
err_eig =

  -0.0069 - 0.0258i
   0.0227 + 0.0651i
  -0.0258 - 0.0794i
   0.0191 + 0.0635i
  -0.0090 - 0.0234i

p_total_a =
   1.0e-10 *
    0.0000    0.0000    0.0000    0.0001    0.0003
    0.0001    0.0001    0.0003    0.0006    0.0023
    0.0011    0.0011    0.0023    0.0045    0.0182
    0.0045    0.0045    0.0091    0.0364    0.1455
    0.0023    0.0023    0.0045    0.0091    0.0364
```

Since the precision of `p_total_a`, which is the average value of δX, is $1 * 10^{-10}$, we can find that the small perturbation has been significantly enlarged by the operations of calculating the eigenvalue.

b) Then we run the test for the single value. We generate 1000 δX values and calculate average. Then we use each δX to calculate the single value of X + δX, and we subtract that value from the single value of X and calculate the average error. Finally we compare these two errors.

```
err_single = [0;0;0;0;0;];
p_total_b = [0 0 0 0 0;
             0 0 0 0 0;
             0 0 0 0 0;
             0 0 0 0 0;
             0 0 0 0 0;];

for b = 1:1000
    pertur_b = eps(x);
    p_total_b = p_total_b + pertur_b;
    err_single = err_single + svd(x + pertur_b) - svd(x);
end
err_single = err_single / 1000
p_total_b = p_total_b / 1000
```

The result is follows:

```
err_single =
   1.0e-10 *
    0.1455
    0.0008
    0.0092
   -0.0011
   -0.0006

p_total_b =
   1.0e-10 *
    0.0000    0.0000    0.0000    0.0001    0.0003
    0.0001    0.0001    0.0003    0.0006    0.0023
    0.0011    0.0011    0.0023    0.0045    0.0182
    0.0045    0.0045    0.0091    0.0364    0.1455
    0.0023    0.0023    0.0045    0.0091    0.0364
```

This time, we can the absolute error of `p_total_b` and `err_single` is very close. Therefore, we can conclude that SVD is a much more stable operation, which does not enlarge the permutation.

Ex .2 - Simple SVD calculations

$$\overset{3\times 4}{X} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix} \qquad XX^T = \begin{pmatrix} 15 & 35 & 10 \\ 35 & 87 & 34 \\ 10 & 34 & 43 \end{pmatrix}$$

$$\det(XX^T - \lambda I) = (15-\lambda)(87-\lambda)(43-\lambda) + 35\cdot 34\cdot 10\cdot 2 - 100(87-\lambda) - 35^2(43-\lambda) - 34^2(15-\lambda) = 0$$

$$\Rightarrow \lambda_1 = 0.76,\ \lambda_2 = 53.54,\ \lambda_3 = 235.70$$

$$\Rightarrow \sigma_1 = 0.87,\ \sigma_2 = 7.32,\ \sigma_3 = 15.35 \Rightarrow \Sigma = \begin{pmatrix} 0.87 & 0 & 0 & 0 \\ 0 & 7.32 & 0 & 0 \\ 0 & 0 & 15.35 & 0 \end{pmatrix}$$

if $\sigma_1 = 0.87$, $u_1 = (0.90, -0.41, 0.12)$,

if $\sigma_2 = 7.32$, $u_2 = (-0.27, -0.35, 0.90)$

if $\sigma_3 = 15.35$, $u_3 = (-0.33, -0.84, -0.43)$

Then $X^TX = \begin{pmatrix} 107 & 32 & 47 & 62 \\ 32 & 40 & 48 & 56 \\ 47 & 48 & 59 & 70 \\ 62 & 56 & 70 & 84 \end{pmatrix}$

if $\sigma_1 = 0.87$, $v_1 = (-0.48, -0.32, -0.05, -0.82)$

if $\sigma_2 = 7.32$, $v_2 = (-0.37, -0.36, -0.75, 0.41)$

if $\sigma_3 = 15.35$ $v_3 = (-0.55, 0.83, -0.13, 0)$

if $\sigma_4 = 0$, $v_4 = (-0.58, -0.28, 0.64, 0.41)$

$$\Rightarrow X = \begin{pmatrix} u_1, u_2, u_3 \end{pmatrix} \Sigma \begin{pmatrix} v_1^T \\ v_2^T \\ v_3^T \\ v_4^T \end{pmatrix}$$

Ex .3 - PCA in Spark

All code and result can be found in `ex3.ipynb`.

1. PCA can help identify which of the columns are principal to the final column y. Thus, Krystor can find columns representing potential output of the sensors without actually knowing what sensor corresponds to what column.
2. We use `sklearn` package in python to calculate the explained variance ratio of the array.

   We can find that column 1, 2 and 3 have a explained variance ratio of 0.48, 0.34 and 0.17. All of the rest columns have a value close to 0. Therefore, we can say n =3.

3. Now we apply `LinearRegression` model in `sklearn` with the input X, the first 3 columns of `sensor1.csv`, Y, the last column of `sensor1.csv`. Then we can calculate the intercept, coefficient and the error of the model. We can get β_i = [10.19226913, 5.26175262, -5.04791382], x_0 = 732.91.
4. We first do PCA to `sensor2.csv` to find the principal columns. And then we apply similar linear model and we have a R^2 score of 0.99. Therefore, we believe `sensor2.csv` also contains the output of the sensors in the electric circuit of Reapor Rich's new cinema.