# 1002 HW 9

gjd9961

December 4th 2021

1. (Noisy measurement) We are interested in measuring a certain quantity modeled by a random variable $\tilde{x}$ with zero mean and unit variance. Our available measurements $\tilde{y} := \tilde{x} + \tilde{z}$ are corrupted by additive random noise, modeled as a random variable $\tilde{z}$ with zero mean and variance $\sigma^2$, which is independent from $\tilde{x}$.

   (a) What is the best linear estimate of $\tilde{x}$ given $\tilde{y} = y$?

   So we can compute this using the general linear estimate formula and then capitalize on the fact that x is a standardized random variable and that since the mean of x and y is 0 then the variance is just the expectation of each respective variable squared. We can also take advantage of the fact that x and z are independent.

$$\begin{aligned}
\arg\min_{\alpha} E((x-\alpha y)^2) &= E(x^2 + \alpha^2 y^2 - 2\alpha yx) \\
&= E(x^2) + \alpha^2 E(y^2) - 2\alpha E(xy) \\
&= E(x^2) + \alpha^2 E(y^2) - 2\alpha E(x(x+z) \\
&= E(x^2) + \alpha^2 E(y^2) - 2\alpha E(x^2) - 2\alpha E(xz) \\
&= E(x^2) + \alpha^2 E(y^2) - 2\alpha E(x^2) - 2\alpha E(xz) \\
&= 1 + \alpha^2 E(y^2) - 2\alpha
\end{aligned} \tag{1}$$

   Taking the derivative with respect to $\alpha$ and setting the equation to 0 we get that the best value for $\alpha$ is:

$$\alpha = \frac{1}{E(y^2)}$$

   So we now must compute the mean and variance of y.

$$E(y) = E(x+z) = E(x) + E(y) = 0 + 0 = 0$$

$$Var(y) = E(y^2) + E^2(y) = E(y^2) = E((x+z)^2) = E(x^2) + E(z^2) + 2E(xy) = 1 + \sigma^2$$

   Therefore the best estimate for alpha is:

$$\alpha = \frac{1}{1+\sigma^2}$$

   And our best linear estimate for $x$ given $y = y$ is:

$$x = \frac{1}{1+\sigma^2} y$$

(b) What is the corresponding mean squared error? The mean square error is given by the following equation, which we plug *alpha* into:

$$MSE = (1 - \rho_{x,y})\sigma_x^2 = 1 - \frac{1}{1 + \sigma^2}$$

(c) What happens to the estimate and the error when $\sigma \to 0$? Explain why this makes sense.

As sigma approaches 0 we our mean square error will approach 0 as well, and we will have no prediction error. This is because we would effectively be eliminating the variance from our noise variable, and since it the noise variable z has mean of 0, thus this would indicate we are taking perfect measurements of x with y, so our prediction would be spot on.

(d) What happens to the estimate and the error when $\sigma \to \infty$? Explain why this makes sense.

When sigma approaches infinity then the mean square error is 1 and the alpha term in our linear estimate becomes 0. Thus our estimate is 0, which happens to be the mean of x. This makes sense as since the variance of the noise variable z is so large, it makes y an unreliable predictor. It better suits us to guess the mean of x in this scenario.

2. (Rufus) Nora's dog Rufus lives in her garden, which is the shaded area in Figure 1. After

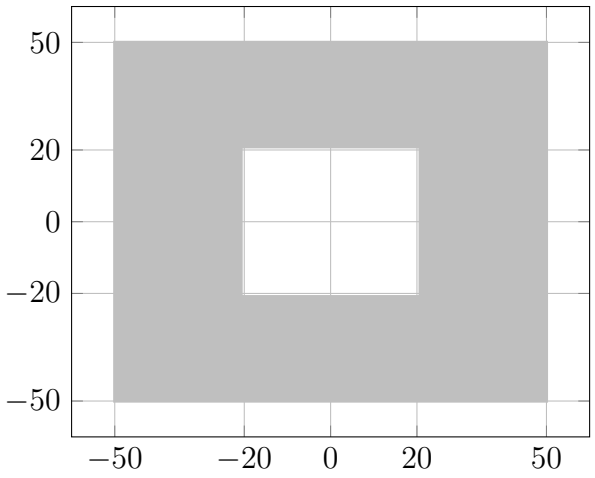



Figure 1: Nora's garden (in gray).

observing Rufus for a while she decides that his position within the garden is uniformly distributed (i.e. the probability density of his position is the same at every point of the garden). Let $\tilde{x}$ be the position of Rufus on the x axis in Figure 1 and $\tilde{y}$ his position on the y axis.

(a) Compute the mean of $\tilde{x}$.

Short answer: the mean of $\tilde{x}$ is 0. We can clearly see this as the uniform distribution that models Rufus' position around the garden is centered about 0 and symmetric. We can compute this more rigorously in the following way once we calculate the pdf of $\tilde{x}$ and $\tilde{y}$.

(b) Compute the pdf of $\tilde{y}$. Sketch it.

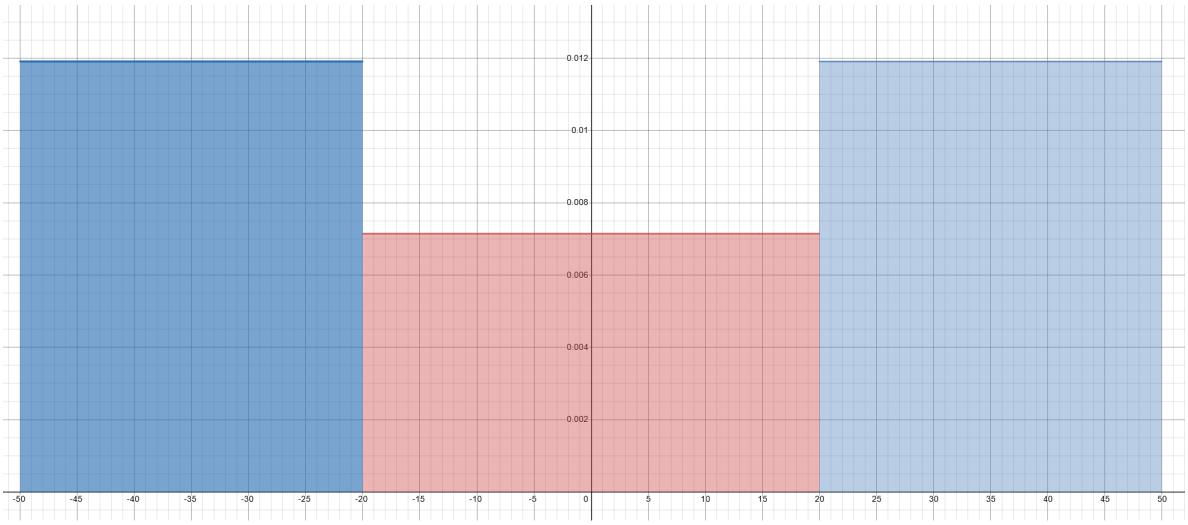
We only need to consider two separate $\tilde{y}$ positions for Rufus, those that are bounded when x is between $[-50, -20]$ (which will work for the position bounded by $[20, 50]$ as the distribution is unifrom and symmetric), and those that are bounded when x is between $[-20, 20]$.
Firstly we calculate the pdf of $\tilde{y}$ for $[-50, -20]$ / $[20, 50]$:

$$\int_{x=-50}^{x=50} f_{\tilde{x},]\tilde{y}}(\tilde{x}, \tilde{y})dx = \int_{x=-50}^{x=50} \frac{1}{8400} = \frac{100}{8400} = \frac{1}{84}$$

Now we calculate the pdf of $\tilde{y}$ for $[-20, 20]$:

$$\int_{x=-50}^{x=20} f_{\tilde{x},]\tilde{y}}(\tilde{x}, \tilde{y})dx = \int_{x=-50}^{x=20} \frac{1}{8400} = 2 * \frac{30}{8400} = \frac{1}{140}$$

Otherwise the pdf is 0. PDF of $\tilde{y}$:



(c) What is the pdf of $\tilde{x}$ conditioned on $\tilde{y}$? Sketch it.

To compute the conditional pdf we'll need to calculate the marginal first. Lets compute it firstly for y in $[-50, -20]$ which since it's symmetric will also work for y in $[20, 50]$:

$$f_{x|y}(x|y) = \frac{f_{x,y}(x,y)}{f_y(y)} = \begin{cases} \frac{1}{100} & \text{when } -50 \leq x \leq 50 \\ 0 & \text{otherwise} \end{cases}$$

Now lets calculate it for y in $[-20, 20]$:

$$f_{x|y}(x|y) = \frac{f_{x,y}(x,y)}{f_y(y)} = \begin{cases} \frac{1}{60} & \text{when } -50 \leq x \leq -20 \text{ or } 20 \leq x \leq 50 \\ 0 & \text{otherwise} \end{cases}$$

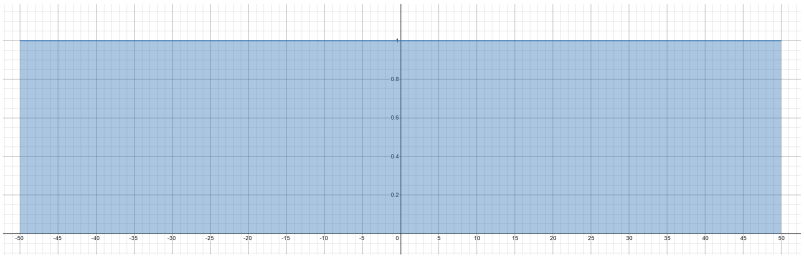


Figure 2: PDF when y in $[-50, -20]$ or $[20, 50]$:

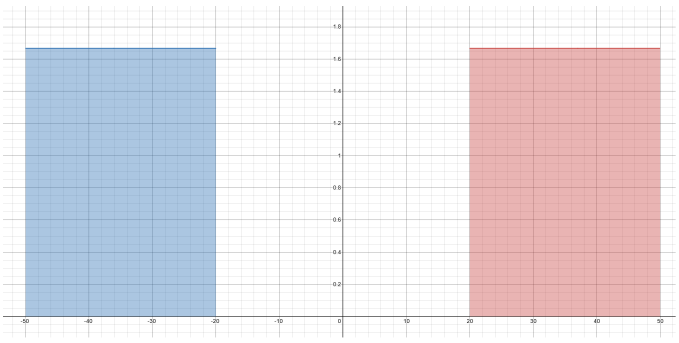


Figure 3: PDF when y in $[-20, 20]$:

(d) Are $\tilde{x}$ and $\tilde{y}$ independent? Justify your answer.

We know that $\tilde{x}$ and $\tilde{y}$ are not independent as the distribution of x and y changes as you find out information about either variable.

$$f_{\tilde{x},\tilde{y}}(0,0) \neq f_{\tilde{x}}(0) f_{\tilde{y}}(0)$$

(e) Are $\tilde{x}$ and $\tilde{y}$ uncorrelated? Justify your answer.

$\tilde{x}$ and $\tilde{y}$ are indeed uncorrelated, which feels counter intuitive but once you analyze the mean and the variance of each random variable it comes clear. The mean of both $\tilde{x}$ and $\tilde{y}$ is 0, as they are centered about 0 and symmetric. Therefore, we only need to compute $E(xy)$ to calculate correlation via iterated expectation:

$$E(xy|y) = \int_{x=-50}^{x=50} \frac{xy}{100} dx = \frac{xy}{100} \int_{x=-50}^{x=50} x dx = 0$$

This also works for the other bound:

$$E(xy|y) = \frac{1}{60} (\int_{x=-50}^{x=-20} x dx + \int_{x=20}^{x=50} x dx) = 0$$

Thus the two random variables are uncorrelated as their covariance is 0.

3. (Random vector) A random vector $\tilde{x}$ with zero mean has a covariance matrix $\Sigma_{\tilde{x}}$ with the following eigendecomposition:

$$\Sigma_{\tilde{x}} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \tag{2}$$

(a) What is the variance of each of the entries of the random vector $\tilde{x}[1]$, $\tilde{x}[2]$ and $\tilde{x}[3]$?

We can calculate the covariance matrix by doing the above matrix factorilization given in the problem statement. When we analyze the diagonal values of the resulting matrix, we'll have the variance of each random variable. When we do we find that the variance of $\tilde{x}[1]$, $\tilde{x}[2]$ and $\tilde{x}[3]$ is equal to $1, 0.25, 0.25$ respectively.

$$\Sigma_{\tilde{x}} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0.25 & 0.25 \\ 0 & 0.25 & 0.25 \end{pmatrix} \tag{3}$$

(b) Is it possible to find a unit-norm vector $u$ such that the inner product between $\tilde{x}$ and $u$ (i.e. the amplitude of the projection of $\tilde{x}$ onto that direction) has variance greater than 1?

Since we are given the eigenvalues and eigenvectors from the covariance matrix eigen decomposition, we can see that the largest eigenvalue is 1. If we take a unit-norm vector and project it via inner product between $x$ and $u$ to achieve maximum variance, we see that the most we can amplify or scale the vector is 1. Therefore, we cannot have a variance greater by 1 as $1 \times 1 = 1$.

(c) Find three constants $a_1$, $a_2$ and $a_3$, such that at least one of them is nonzero and $P(a_1\tilde{x}[1] + a_2\tilde{x}[2] + a_3\tilde{x}[3] = 0) = 1$. Justify your answer mathematically, and interpret it geometrically.

This problem is bascially asking "can we create the 0 vector for some (non-zero) linear combination of our random variables" which is the same as calculating a null space in linear algebra. Since random variables 2 and 3 are correlated 1:1 which I will show below, we can combine the two in a linear combination such that the result is alawys the 0 vector (this includes random variable 1 with scalar 0). Therefore the answer is: $a_1 = 0, a_2 = 1, a_3 = -1$

$$P(0 \times \tilde{x}[1] + 1 \times \tilde{x}[2] + -1 \times \tilde{x}[3] = 0) = 1$$

We know this will work because $\tilde{x}[2]$ and $\tilde{x}[3]$ are correlated 1:1 as shown below:

$$Corr_{2,3} = \frac{cov(a,b)}{\sigma_a \sigma_b} = \frac{.25}{.5 \times .5} = 1$$

4. (Financial data) In this exercise you will use the code in the findata folder. For the data loading code to work properly, make sure you have the pandas Python package installed on your system.

Throughout, we will be using the data obtained by calling `load_data()` in `findata_tools.py`. This will give you the names, and closing prices for a set of 18 stocks over a period of 433 days ordered chronologically. For a fixed stock (such as msft), let $P_1, \ldots, P_{433}$ denote its sequence of closing prices ordered in time. For that stock, define the daily returns series $R_i := P_{i+1} - P_i$ for $i = 1, \ldots, 432$. Throughout we think of the daily stock returns as features, and each day (but the last) as a separate datapoint in $^{18}$. That is, we have 432 datapoints each having 18 features.

(a) Looking at the first two principal directions of the centered data, give the two stocks with the largest coefficients (in absolute value) in each direction. Give a hypothesis why these two stocks have the largest coefficients, and confirm your hypothesis using the data. The file `findata_tools.py` has `pretty_print()` functions that can help you output your results. You are not required to include the principal directions in your submission.

In the first two principle components, Amazon and Google both have the highest weighted. This is because they each have large variance, but also have large share prices so they generate larger daily returns, thus skewing the weightings of the variance.

(b) Standardize the centered data so that each stock (feature) has variance 1 and compute the first 2 principal directions. This is equivalent to computing the principal directions of the correlation matrix (the previous part used the covariance matrix). Using the information in the comments of *generate_findata.py* as a guide to the stocks, give an English interpretation of the first 2 principal directions computed here. You are not required to include the principal directions in your submission.

After we center / standardize the data, and calculate the principle components, our first two PCs are much more interesting. The first principle component applies heavy weighting to the following stocks: SPY (S&P 500 ETF) which indicates its variance is driven by the variance of the overall stock market, and its returns are generated by market movements. The second principle component applies heavy weighting to the following stock: USO, which is oil commodities. We can interpret the two principle components in the following way: when investors are bullish, the market rises, hence the first principle component. When investors are bearish, the stock market sinks, and oil which is seen as a hedge, and tends to spike when market conditions are subpar, rises.

(c) Assume the stock returns each day are drawn independently from a multivariate distribution $\tilde{x}$ where $\tilde{x}[i]$ corresponds to the $i$th stock. Assume further that you hold a portfolio with 200 shares of each of appl, amzn, msft, and goog, and 100 shares of each of the remaining 14 stocks in the dataset. Using the sample covariance matrix as an estimator for the true covariance of $\tilde{x}$, approximate the standard deviation of your 1 day portfolio returns $\tilde{y}$ (this is a measure of the risk of your portfolio). Here $\tilde{y}$ is given by

$$\tilde{y} := \sum_{i=1}^{18} \alpha[i]\tilde{x}[i],$$

where $\alpha[i]$ is the number of shares you hold of stock $i$.

$$\tilde{y} := \sum_{i=1}^{18} \alpha[i]\tilde{x}[i] = 6962$$

(d) Assume further that $\tilde{x}$ from the previous part has a multivariate Gaussian distribution. Compute the probability of losing 1000 or more dollars in a single day. That

is, compute

$$\Pr(\tilde{y} \leq -1000).$$

The expected daily return from our portfolio is 879.78. Since we have the variance and mean of our porfolio, we can now calculate the probability of losing more than 1000 a day using a normal distribution. The probability is approximately 39.36%

Note: The assumptions made in the previous parts are often invalid and can lead to inaccurate risk calculations in real financial situations.

# Table of Contents



## 1 Import Libraries and Given Functions

In [1]:
```
from __future__ import print_function
import pandas as pd
import numpy as np
from numpy import linalg

"""
Loads financial data as a pandas dataframe
"""
```

executed in 810ms, finished 15:54:53 2021-12-04

Out[1]:
```
'\nLoads financial data as a pandas dataframe\n'
```

In [2]:

```
def load_dataframe(filename):
    return pd.read_csv(filename, index_col=0)


"""
Loads financial data as a tuple: names,data.
names is a list of the stock names represented in each column.
data is a 2d numpy array.  Each row of data corresponds to a trading day.
data[i,j] is the price (technically the adjusted closing price) of
instrument names[j] on the ith day.  The days are ordered chronologically.
"""


def load_data(filename):↔


"""
Given a 1d numpy array vec of n values, and a list of n names,
prints the values and their associated names.
"""


def pretty_print(vec, names):↔


"""
Given a 1d numpy array vec of n values, and a list of n names,
prints the values and their associated names in a LaTeX friendly
format.
"""


def pretty_print_latex(vec, names, num_col=6):↔


def main():↔
    # pretty_print_latex(data[0,:],names)


if __name__ == "__main__":↔
```

executed in 60ms, finished 15:54:53 2021-12-04

```
# of stocks = 18, # of days = 433
      AAPL    AMZN     MSFT    GOOG     XOM      APC      CVX        C  \
  112.6208  753.67  60.1715  786.14  83.316  68.7428  109.136  58.0193

        GS      JPM       AET       JNJ      DGX       SPY      XLF       SS
O  \
  235.3232  82.7023  120.0301  109.9246  88.5165  216.8377  22.7081  76.400
6

      SDS    USO
  58.5837  11.44
```

In [3]:
```
names, df = load_data('stockprices.csv')
df = load_dataframe('stockprices.csv')
```
executed in 29ms, finished 15:54:54 2021-12-04

In [4]:
```
len_df = len(df)
df1 = df.iloc[0:len(df)-1, :]
df2 = df.iloc[1:len(df), :]
df3 = df2.reset_index(drop=True) - df1.reset_index(drop=True)
print(df2.shape, df1.shape)
```
executed in 44ms, finished 15:54:54 2021-12-04

```
(432, 18) (432, 18)
```

## 2 PCA Function Below

In [5]:

```
def pca(data, stand=False, k=None, var=False):

    cols = list(data.iloc[:0])
    data_copy = data.copy
    # Center Data at 0, Each Column must have mean 0

    # Loop through every column
    data_copy = data - data.mean()

    # If we should standardize, then standardize the dataset
    if stand == True:
        data_copy = data_copy / data_copy.std()

    # Compute Covariance Matrix
    Cov_matrix = data_copy.T @ data_copy

    # Calculate eigendecomp for Cov_Matrix
    vals, vectors = np.linalg.eigh(Cov_matrix)

    # Sort the eigenvalues descending order
    vals = vals[::-1]
    vectors = vectors[:, ::-1]

    # If we wanted the least dimensions for a certain amount of variance ex

    # See if Var was passed to function and its valid
    if var > 0 and var <= 1:
        tracker = 0
        eig_vals_of_interest, eig_vectors_of_interest = [], []
        total_var = vals.sum()
        for i in range(len(vals)):
            if tracker < var:
                tracker += vals[i] / total_var
                eig_vals_of_interest.append(vals[i])
        eig_vectors_of_interest = vectors[:, :len(eig_vals_of_interest)]

    # Check if we just wanted k dimensions
    elif k > 0 and k <= len(cols):

        # If we wanted k dimensions, set the appropriate amount
        eig_vals_of_interest = vals[:k]
        eig_vectors_of_interest = vectors[:, :k]
    else:
        return vals, vectors

    # Make sure data types are compatible
    eig_vectors_of_interest = np.array(eig_vectors_of_interest)

    # Compute the data projected onto the pcas
    new_data = data_copy@eig_vectors_of_interest

    #Returns the data projected onto the pcas, the eigenvalues, and the eig
    return Cov_matrix, eig_vals_of_interest, eig_vectors_of_interest
```

executed in 14ms, finished 15:54:54 2021-12-04

## 3 Prolem A

In [6]:
```
new, vals, vec = pca(df3, stand=False, k=18)
print('PCA 1: \n')
pretty_print(vec[:, 0], names)
print('\n','PCA 2: \n')
pretty_print(vec[:, 1], names)
```

executed in 93ms, finished 15:54:54 2021-12-04

```
PCA 1:

     AAPL     AMZN      MSFT      GOOG       XOM       APC       CVX  \
-0.054553 -0.86793 -0.036651 -0.482672 -0.007916 -0.009795 -0.013876

        C        GS       JPM       AET       JNJ       DGX       SPY  \
-0.012407 -0.053403 -0.020728 -0.008575 -0.013319 -0.011993 -0.054358

      XLF       SSO       SDS       USO
-0.004979 -0.044236  0.016887 -0.001485

 PCA 2:

     AAPL      AMZN      MSFT      GOOG       XOM       APC      CVX  \
 0.041894 -0.494995  0.026756  0.851087  0.028004  0.009165  0.02881

       C        GS       JPM       AET       JNJ       DGX       SPY  \
 0.02591  0.114993  0.037115  0.028005  0.041139  0.011263  0.069472

      XLF       SSO       SDS       USO
 0.009083  0.056721 -0.021421  0.000401
```

## 4 Problem B

In [7]:
```
new1, vals, vec = pca(df3, k=18)
print('PCA 1: \n')
pretty_print(vec[:, 0], names)
print('\n','PCA 2: \n')
pretty_print(vec[:, 1], names)
```

executed in 42ms, finished 15:54:54 2021-12-04

```
PCA 1:

     AAPL      AMZN      MSFT      GOOG       XOM       APC       CVX  \
-0.054553 -0.86793 -0.036651 -0.482672 -0.007916 -0.009795 -0.013876

        C        GS       JPM       AET       JNJ       DGX       SPY  \
-0.012407 -0.053403 -0.020728 -0.008575 -0.013319 -0.011993 -0.054358

      XLF       SSO       SDS       USO
-0.004979 -0.044236  0.016887 -0.001485

 PCA 2:

     AAPL      AMZN      MSFT      GOOG       XOM       APC      CVX  \
 0.041894 -0.494995  0.026756  0.851087  0.028004  0.009165  0.02881

       C        GS       JPM       AET       JNJ       DGX       SPY  \
 0.02591  0.114993  0.037115  0.028005  0.041139  0.011263  0.069472

      XLF       SSO       SDS       USO
 0.009083  0.056721 -0.021421  0.000401
```

## 5 Problem C ¶

In [8]:
```
shares_1 = [200]*4
shares_2 = [100]*14
shares = shares_1 + shares_2
shares = np.array(shares)
```

executed in 12ms, finished 15:54:54 2021-12-04

In [9]:
```

new1 = new1 / 431
number1 = shares.T @ new1
number1 = number1 @ shares
```

executed in 13ms, finished 15:54:54 2021-12-04

In [22]:
```
std_dev = number1 **.5
print(std_dev)
```

executed in 17ms, finished 16:04:45 2021-12-04

```
6962.072274462166
```

## 6 Problem D

In [17]:
```
#Initialize variables
list_of_means = []

#Iterate through the columns and get the mean return
for i in range(len(df3.iloc[0,:])):
    list_of_means.append(np.mean(df3.iloc[:,i]))

#Calculate expected value
expected_value = shares.T @ list_of_means
```

executed in 12ms, finished 15:59:50 2021-12-04

In [18]:
```
#Print expected value of our portfolio
print(expected_value)
```

executed in 14ms, finished 15:59:50 2021-12-04

```
879.7824537037035
```

In [23]:
```
#Calculate Z-Score
z_score = (-1000 - expected_value) / std_dev
```

executed in 16ms, finished 16:04:58 2021-12-04

In [24]:
```
z_score
```

executed in 13ms, finished 16:04:59 2021-12-04

Out[24]:
```
-0.270003294938348
```

In [ ]: