Homework 4 Quantitative Risk Management

Group G03
Jakob Amaya Scott, Martin Hammerues, Albin Henriksson, Markus Trætli

October 2022

Question 1

Since W follows a discrete distribution one may use the law of total probability to find the cumulative distribution of X. The cumulative distribution is

$$F_X(x) = P(X \leq x) = P(\mu + \sqrt{W}Z \leq x) = \sum_{i=1}^{n} P(\mu + \sqrt{k_i}Z \leq x \mid W = k_i)P(W = k_i) =$$

$$= \sum_{i=1}^{n} P(Z \leq \frac{x-\mu}{\sqrt{k_i}})p_i = \sum_{i=1}^{n} \Phi(\frac{x-\mu}{\sqrt{k_i}})p_i.$$

By definition of $VaR_\alpha(X)$, clearly $F_X(VaR_\alpha(X)) = \alpha$, and plugging in $x = VaR_\alpha(X)$ into the last expression gives

$$\sum_{i=1}^{n} \Phi(\frac{VaR_\alpha(X)-\mu}{\sqrt{k_i}})p_i = \alpha.$$

For this reason, we can pick the function

$$f(v) = F_X(v) - \alpha.$$

To see that the root of f is unique, we first notice that $-\alpha < f(v) < 1 - \alpha$ since $F_X(v)$ is between 0 and 1 due to the nature of CDF's. Now since the CDF of the normal distribution is strictly increasing, so is linear combinations of normal CDF's where all coefficients are positive (all $p_i > 0$). We've already established that f is bounded, thus we can conclude that the root $v_0 = VaR_\alpha(X)$ is unique.

It is however not necessary to know the CDF of X to solve this problem. We could've simply picked

$$f(v) = v - VaR_\alpha(X)$$

as our function. It is easy to see that it has a unique root.

Question 2

Consider a random variable X with the following probability mass function:

$$P(X = x) = \begin{cases} 0.25, \text{ if } x = -2, \\ 0.25, \text{ if } x = -1, \\ 0.25, \text{ if } x = 1, \\ 0.25, \text{ if } x = 2, \end{cases}$$

Now consider the new random variable $Y = X^2$. Clearly, X and Y are not independent. The correlation between X and Y, ρ_{XY} is defined as

$$\rho_{XY} = \frac{\mathbb{E}[XY] - \mathbb{E}[X]\,\mathbb{E}[Y]}{\sqrt{Var(X)Var(Y)}}.$$

We have

$$\mathbb{E}[XY] = 0.25(1 \cdot 1^2 + (-1) \cdot (-1)^2 + 2 \cdot 2^2 + (-2)(-2)^2 = 0.$$

Now since $\mathbb{E}[X] = 0$, it follows that the correlation between X and Y is 0.

Question 3

3-factor model

In this model, we used the SPY, VXX and GLD tickers as our factors. The 800-900 datapoints that included NaN's were removed, reducing the number of dates considered to 2675. The rolling VaR and realized returns can be seen in figure 1. The total number of times where the realized loss was greater than VaR at level 95% and 99% was 209 and 151 respectively. We will use the same binomial argument as in the last assignment, that is, rejecting the hypothesis that the loss follows this model if the p-value

$$p = 1 - BinCDF(209, 2425, \alpha)$$

is too low. For $\alpha = 0.95$ and $\alpha = 0.99$, we get $p-$values of $5.8 \cdot 10^{-14}$ and 0 respectively. In other words, this model is not a good one.

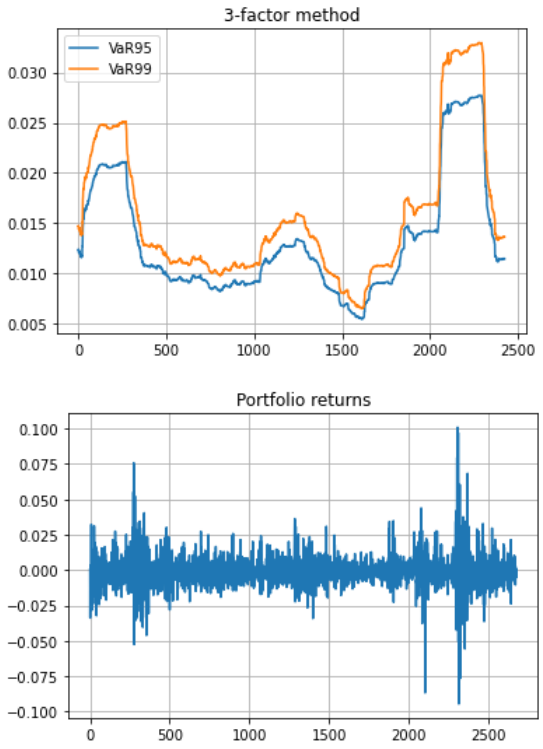

Figure 1: VaR at the given significance levels, and realized returns for the 3-factor model.

5 component PCA model

This time, we have 3603 datapoints. Instead of using the 3 tickers above as factors, we will now use the top 5 PCA components. This results in the VaR graphs seen in figure 2. The total number of times where the realized loss was greater than VaR at level 95% and 99% was 276 and 205 respectively. Again, using the $p-$value test, for $\alpha = 0.95$ and $\alpha = 0.99$, we obtain $p-$values of $1.9 \cdot 10^{-15}$ and 0 respectively.

It is worth noting that even though these two factor models are not good, they are at least better than the model used in the last assignment.

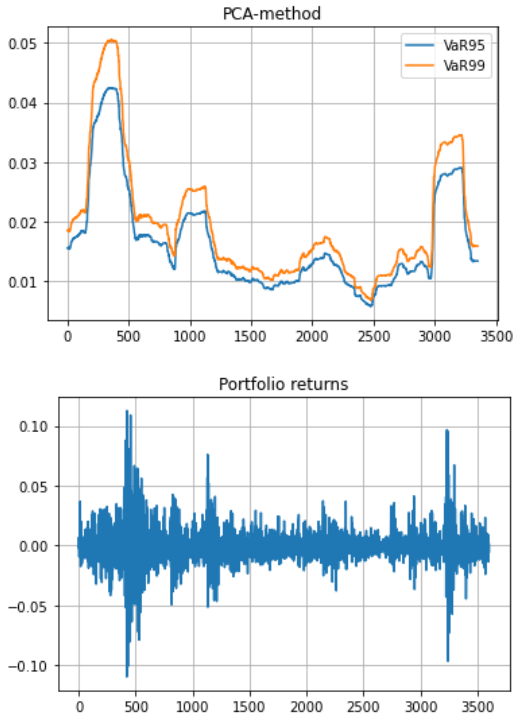

Figure 2: VaR at the given significance levels, and realized returns for the PCA model.

Question 4

a)

Based on 10 000 samples of the portfolio, the estimated VaR_α for significance level $\alpha = 0.95$ was found to be

$$VaR_{0.95} \approx 10.4164.$$

b)

The eigenvector and eigenvalue of the first principal component of X was found to be

$$\vec{\mathbf{v}_1} = \begin{bmatrix} 0.07525 \\ -0.06627 \\ -0.9949 \\ 0.00342 \end{bmatrix} \quad \text{and} \quad \lambda_1 = 22.6099.$$

In terms of magnitude, the third component of the eigenvector is clearly largest, and therefore captures most of the variance in this component. The estimated covariance matrix $\overline{\Sigma}$ of X was found to be

$$\overline{\mathbf{\Sigma}} = \begin{bmatrix} 1.7334 & 1.7423 & -1.7396 & 1.7263 \\ 1.7423 & 3.4370 & 1.6660 & 0.0868 \\ -1.7396 & 1.6660 & \mathbf{23.6165} & -0.1339 \\ 1.7262 & 0.0868 & -0.1339 & 6.6512 \end{bmatrix}.$$

In the covariance matrix one can see that the variance of the third component is much larger than for the other components. Therefore it is natural to assume that this component would be able to explain most of the variance in the distribution, and subsequently it will play a big part in the first principal component.

c)

The empirical principal component transform for component k is given by

$$\mathbf{Y}_k = \overline{\mathbf{\Gamma}}^T * (\mathbf{X}_k - \overline{\boldsymbol{\mu}}).$$

Then define $\mathbf{Y}$ by

$$\mathbf{Y} = \left[\mathbf{Y}_1, \mathbf{Y}_2, ..., \mathbf{Y}_d\right]$$

where d is the number of components. In this case $d = 4$. The principal components transform is

$$\mathbf{Y} = \mathbf{\Gamma}^\top (\mathbf{X} - \boldsymbol{\mu}),$$

which can be inverted as follows

$$\mathbf{X} = \boldsymbol{\mu} + \mathbf{\Gamma}\mathbf{Y}.$$

By partitioning $\mathbf{Y}$ and $\mathbf{\Gamma}$ into

$$\mathbf{Y} = [\mathbf{Y}_1^\top, \mathbf{Y}_2^\top]^\top \text{ and } \mathbf{\Gamma} = [\mathbf{\Gamma}_1, \mathbf{\Gamma}_2],$$
$$\text{where } \mathbf{Y}_1 \in \mathbb{R}^k, \mathbf{Y}_2 \in \mathbb{R}^{d-k}, \mathbf{\Gamma}_1 \in \mathbb{R}^{d\times k}, \text{ and } \mathbf{\Gamma}_2 \in \mathbb{R}^{d\times(d-k)}.$$

Here $k = 2$ is the number of principal components we want to use in our approximation. This gives

$$\mathbf{X} = \boldsymbol{\mu} + \mathbf{\Gamma}_1\mathbf{Y}_1 + \mathbf{\Gamma}_2\mathbf{Y}_2 + \boldsymbol{\varepsilon} \approx \boldsymbol{\mu} + \mathbf{\Gamma}_1\mathbf{Y}_1.$$

By this method the estimated value of $VaR_{0.95}$ was $VaR_{0.95} \approx 9.8716$. The approximation is not too far away from our previous estimate, and is slightly smaller. From the approximation, we only considered the first two principal components, and neglected the last two. In practice this means that our estimate of the distribution has neglected some of the variance, which naturally means that the variance will be lower. As a consequence, the VaR_α will be smaller.

Code

Q.3

3-factor model

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm


alpha = .95
T = 250
theta = 1

df = pd.read_csv('stock_prices_SP500_2007_2021.csv')
factors = df[['SPY', 'VXX', 'GLD']]
stocks = df.drop(columns=['time'])
stocks = stocks.dropna(axis=1)  # Drop stocks with NaN-values
regressors = pd.concat([factors, stocks], axis=1,
                       sort=False)  # Create a matrix with all factors and stocks. The first 3 columns are the independent variables in the regression model.
regressors = regressors.dropna(
    axis=0)  # Drop all days with any NaN values. This removes about 1/4 of the data in the beginning of the period
diff = np.zeros((len(regressors.iloc[:, 0]) - 1, len(regressors.iloc[0, :])))

q = np.random.rand(len(stocks.iloc[0, :]), 1)
q = q / np.sum(q)  # Random portfolio where the elements sum to 1.

"""Calculates loss"""
for i in range(len(regressors.iloc[:, 0]) - 1):
    diff[i, :] = (np.log(regressors.iloc[i, :]) - np.log(regressors.iloc[i + 1, :]))

stocks_diff = diff[:, 3:]
factors_diff = diff[:, 0:3]
VaR = []

"""Calculates VaR. Formulas are largely obtained from page 22 onwards in the lectures notes of lecture 4."""

for i in range(len(diff[:, 0]) - T):
    omega = np.cov(factors_diff[i:i + T, :].T)
    F = np.hstack((np.ones((T, 1)), factors_diff[i:i + T, :]))
    B = np.linalg.inv(F.T @ F) @ F.T @ stocks_diff[i:i + T, :]  # OLS estimate of regressors
    a = B[0, :].T
    beta = B[1:, :].T
    residual = stocks_diff[i:i + T, :] - F @ B
    res_cov = np.cov(residual.T)
    res_cov = np.diagonal(res_cov) * np.identity(
        len(res_cov[:, 0]))  # We only want diagonal elements according to the assignment
    portfolio_sigma = beta @ omega @ beta.T + res_cov
    mean = np.mean(stocks_diff[i:i + T, :], axis=0)
    VaR.append((norm.ppf(alpha) * q.T @ portfolio_sigma @ q) ** .5)  # Calculates VaR as on page 48 of lecture 3

VaR = np.array(VaR)
```

```
plt.plot(VaR[:, 0, 0])
plt.show()

"""Calculates number of breaches"""

portfolio_loss=stocks_diff@q
VaR=np.array(VaR)
VaR=VaR.reshape(-1,1)
portfolio_loss=np.array(portfolio_loss)
breaches=portfolio_loss[T:,:]>VaR
print('The number of breaches is {0}'.format(np.sum(breaches)))
```

PCA model

```
import pandas as pd
import random
import sys
import scipy
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
df= pd.read_csv('stock_prices_SP500_2007_2021.csv')
pd.set_option('display.max_rows',None)
pd.set_option('display.max_columns',None)

T = 250
alpha1 = .95
alpha2 = .99
stocks = df.drop(columns=['time'])
stocks = stocks.dropna(axis=1)  # Drop stocks with NaN-values
diff = np.zeros((len(stocks.iloc[:, 0]) - 1, len(stocks.iloc[0, :])))
no_factors = 5
q = np.random.rand(len(stocks.iloc[0, :]), 1)
q = q / np.sum(q)  # Random portfolio where the elements sum to 1.

"""Calculate loss"""
for i in range(len(stocks.iloc[:, 0]) - 1):
    diff[i, :] = (np.log(stocks.iloc[i, :]) - np.log(stocks.iloc[i + 1, :]))
VaR1 = []
VaR2 = []

"""Calculate VaR based on the factor approach with top 5 PCA elements as factors."""

for i in range(len(diff[:, 0]) - T):
    cov_matrix = np.cov(diff[i:T + i, :].T)
    eigenValues, eigenVectors = np.linalg.eig(cov_matrix)
    idx = eigenValues.argsort()[::-1]
    eigenValues = eigenValues[idx]
    eigenVectors = eigenVectors[:, idx]  # This matrix turns out to be orthonormal
    factors_diff = diff[i:i + T] @ eigenVectors[:, 0:no_factors]  # Factors are the
        principal components

    """From here, we do the same as in the previous part."""
    omega = np.cov(factors_diff.T)
    F = np.hstack((np.ones((T, 1)), factors_diff))
    B = np.linalg.inv(F.T @ F) @ F.T @ diff[i:i + T, :]  # OLS estimate of regressors
    a = B[0, :].T
    beta = B[1:, :].T
    residual = diff[i:i + T, :] - F @ B
    res_cov = np.cov(residual.T)
    res_cov = np.diagonal(res_cov) * np.identity(
```

```
        len(res_cov[:, 0])) # We only want diagonal elements according to the assignment
    portfolio_sigma = beta @ omega @ beta.T + res_cov
    mean = np.mean(diff[i:i + T, :], axis=0)
    VaR1.append((norm.ppf(alpha1) * q.T @ portfolio_sigma @ q) ** .5) # Calculates VaR as on page 48 of lecture 3
    VaR2.append((norm.ppf(alpha2) * q.T @ portfolio_sigma @ q) ** .5) # Calculates VaR as on page 48 of lecture 3

VaR1 = np.array(VaR1)
VaR2 = np.array(VaR2)
plt.title('PCA-method')
plt.plot(VaR1[:, 0, 0], label='VaR95')
plt.plot(VaR2[:, 0, 0], label='VaR99')
plt.legend()
plt.grid(True)
plt.show()
plt.title('Portfolio returns')
plt.plot(diff @ q)
plt.grid(True)
plt.show()


"""Calculates number of breaches"""

portfolio_loss=diff@q
VaR1=np.array(VaR1)
VaR1=VaR1.reshape(-1,1)
portfolio_loss=np.array(portfolio_loss)
breaches=portfolio_loss[T:,:]>VaR1
print(np.sum(breaches))
print(np.sum(breaches)/len(VaR1))


VaR2=np.array(VaR2)
VaR2=VaR2.reshape(-1,1)
portfolio_loss=np.array(portfolio_loss)
breaches=portfolio_loss[T:,:]>VaR2
print(np.sum(breaches))
print(np.sum(breaches)/len(VaR2))
```

Q.4

```
import numpy as np
import pandas as pd
import numpy.random



N = 10000
A = np.matrix([[1, 0, 0, 0],
    [1, 1, 0, 0],
    [-1, 2, 3, 0],
    [1, -1, 1, 1]])

x = numpy.random.standard_t(df = 5, size = (N, 4))
X = pd.DataFrame((A@(x.T)).T)
L = np.sum(X, axis = 1)

# Quesiton 1
VaR_095 = np.sort(L, axis = 0)[int(N*0.95)]
print("Value-at-Risk: ",VaR_095)
```

```
20
21 #Question 2
22 C = np.cov(X, rowvar=False)
23 evals, evecs = np.linalg.eig(C)
24 idx = np.argsort(evals)[::-1]
25 evecs = evecs[:,idx]
26 evals = evals[idx]
27 print("First PC:")
28 print("Eigenvector:",evecs[:,0])
29 print("Eigenvalue:", evals[0])
30 print("Covariance matrix:\n", np.cov(X.T))
31
32 #Question 3
33 mu = np.mean(X.to_numpy())
34 Y = evecs.T@(X.to_numpy().T - mu) #estimate Y
35
36 X_approx =  mu + evecs[:,0:2]@Y[0:2,:] #Use the two first principal components with Y
37 L_approx = np.sum(X_approx, axis = 0)
38
39 VaR_095 = np.sort(L_approx, axis = 0)[int(N*0.95)]
40 print("Value-at-Risk: ",VaR_095)
```