Quantitative Risk Management - Assignment

Lukas Schreiner & Giuliano Cunti

December 3, 2018

Contents

1 Question I

1.1 Model 2

The return vector R is bivariate Gaussian distributed with mean vector $\mu$and covariance matrix Σ.

In this particular model, the density function of the bivariate Gaussian distribution is given as follows:

$$f(x_1, x_2; \mu, \Sigma) = \frac{1}{2\pi\sigma_1\sigma_2\sqrt{(1-\rho^2)}} e^{-\frac{1}{2(1-\rho^2)}(\frac{(x_1-\mu_1)^2}{\sigma_1^2} - \frac{2\rho(x_1-\mu_1)(x_2-\mu_2)}{\sigma_1\sigma_2} + \frac{(x_2-\mu_2)^2}{\sigma_2^2})},$$

where (x_1, x_2) are the values of the return vector R of the two portfolios given.

The exercise requires to find an maximum likelihood estimator for the distribution means and the variances. First, the maximum likelihood function is defined.

$$L = \prod_{t=1}^{500} f(x_1, x_2; \mu, \Sigma) = (\frac{1}{2\pi\sigma_1\sigma_2\sqrt{(1-\rho^2)}})^{500} e^{-\frac{1}{2(1-\rho^2)} \sum_{t=1}^{500} (\frac{(x_1-\mu_1)^2}{\sigma_1^2} - \frac{2\rho(x_1-\mu_1)(x_2-\mu_2)}{\sigma_1\sigma_2} + \frac{(x_2-\mu_2)^2}{\sigma_2^2})})$$

Using MLE, we can make use of a log transformation in order to simplify the likelihood function.

$$logL = ln((\frac{1}{2\pi\sigma_1\sigma_2\sqrt{(1-\rho^2)}})^{500} e^{-\frac{1}{2(1-\rho^2)} \sum_{t=1}^{500} (\frac{(x_1-\mu_1)^2}{\sigma_1^2} - \frac{2\rho(x_1-\mu_1)(x_2-\mu_2)}{\sigma_1\sigma_2} + \frac{(x_2-\mu_2)^2}{\sigma_2^2})})$$

Rewriting the equation yields:

$$logL = -500(ln(2) + ln(\pi) + ln(\sigma_1) + ln(\sigma_2) + \frac{1}{2}ln(1-\rho^2)) - \frac{1}{2(1-\rho^2)}(...)$$
$$(...) \sum_{t=1}^{500} \left(\frac{(x_1-\mu_1)^2}{\sigma_1^2} - \frac{2\rho(x_1-\mu_1)(x_2-\mu_2)}{\sigma_1\sigma_2} + \frac{(x_2-\mu_2)^2}{\sigma_2^2} \right)$$

As a next step, the transformed likelihood function is derivated with respect to the parameters of interest and set equal to 0 (maximization problem). The first order conditions thus are

$$\frac{\partial logL}{\partial \mu_1} = 0$$

$$\frac{\partial logL}{\partial \mu_2} = 0$$

$$\frac{\partial logL}{\partial \sigma_1} = 0$$

$$\frac{\partial logL}{\partial \sigma_2} = 0$$

$$\frac{\partial logL}{\partial \rho} = 0$$

Alternatively, this is equivalent to minimizig g(μ) $= \sum_t^n (\mathrm{X}_i - \mu)^2$in the case of the means. However, derive the following derivatives in order to solve the maximization problem.

$$\frac{\partial logL}{\partial \mu_1} = -\frac{1}{2(1-\rho^2)} \sum_{t=1}^{500} \left(\frac{-2(x_1 - \mu_1)}{\sigma_1^2} - \frac{2\rho(x_2 - \mu_2)}{\sigma_1 \sigma_2} \right) = 0$$

$$\frac{\partial logL}{\partial \mu_2} = -\frac{1}{2(1-\rho^2)} \sum_{t=1}^{500} \left(-\frac{2\rho(x_1 - \mu_1}{\sigma_1 \sigma_2} - \frac{2(x_2 - \mu_2)}{\sigma_2^2} \right) = 0$$

$$\frac{\partial logL}{\partial \sigma_1} = \frac{-500}{\sigma_1} - \frac{1}{2(1-\rho^2)} \sum_{t=1}^{500} \left(-\frac{(x_1 - \mu_1)^2}{\sigma_1^3} + \frac{2\rho(x_1 - \mu_1)(x_2 - \mu_2)}{\sigma_1} \right) = 0$$

$$\frac{\partial logL}{\partial \sigma_2} = \frac{-500}{\sigma_2} - \frac{1}{2(1-\rho^2)} \sum_{t=1}^{500} \left(-\frac{(x_2 - \mu_2)^2}{\sigma_2^3} + \frac{2\rho(x_1 - \mu_1)(x_2 - \mu_2)}{\sigma_2} \right) = 0$$

$$\frac{\partial logL}{\partial \rho} = \frac{-500\rho}{(1-\rho^2)} - \frac{\rho}{(1-\rho^2)^2} \sum_{t=1}^{500} \left(\frac{(x_1 - \mu_1)^2}{\sigma_1^2} - \frac{2(x_1 - \mu_1)(x_2 - \mu_2)}{\sigma_1 \sigma_2} + \frac{(x_2 - \mu_2)^2}{\sigma_2^2} \right) = 0$$

Using statistical packages in Python we derived the respective solutions. We can draw on the fact that for the gaussian distribution, the MLE of the population means and variances are equal to the sample means and sample variances[1].

$$\hat{\mu}_1 = 0.0002768$$

$$\hat{\mu}_2 = 0.0005886$$

$$\hat{\sum} = \begin{pmatrix} \hat{\sigma}_1^2 & \hat{\rho}\hat{\sigma}_1\hat{\sigma}_2 \\ \hat{\rho}\hat{\sigma}_1\hat{\sigma}_2 & \hat{\sigma}_2^2 \end{pmatrix} = \begin{pmatrix} 3.88e-05 & 1.88e-05 \\ 1.88e-05 & 4.22e-05 \end{pmatrix}$$

$$\hat{\sigma}_1 = 0.0062363$$

$$\hat{\sigma}_2 = 0.0064969$$

$$\hat{\rho} = \frac{Cov}{\sigma_1 \sigma_2} = 0.465021$$

$\hat{\sum}$ denotes the MLE of the given covariance matrix.

[1]All results are rounded to 6 decimals

1.2 Model 3

R_i is Gaussian distributed with mean μ and variance σ, for i = 1, 2. These parameters are the same as for model M2. Moreover, the normalized vector of returns $\left(\Phi(\frac{R_1-\mu_1}{\sigma_1}), \Phi(\frac{R_2-\mu_2}{\sigma_2})\right)'$ possesses a Gumbel-copula with parameter $\theta \geq 1$.

First, the data is standardized by subtracting the mean estimated in M2 and dividing by the estimated standard deviation. This is exemplified by the following Gaussian marginal

$$F_X(x_i) = \frac{1}{\sqrt{2\pi}\hat{\sigma}_i} e^{-\frac{1}{2}(\frac{(x_i-\hat{\mu}_i)}{\hat{\sigma}_i})^2}$$

Drawing on the formulae given in the exercise, these standardized values are plugged into the Gumbel-copula function. Thereby the explicit formula is obtained.

$$C(u_1, u_2) = exp(-((-ln(u_1))^\theta + (-ln(u_2))^\theta)^{\frac{1}{\theta}})$$

$\iff$

$$C(u_1, u_2) = exp(-((-ln(\frac{1}{\sqrt{2\pi}\hat{\sigma}_1} e^{-\frac{1}{2}(\frac{(x_1-\hat{\mu}_1)}{\hat{\sigma}_1})^2}))^\theta + (-ln(\frac{1}{\sqrt{2\pi}\hat{\sigma}_2} e^{-\frac{1}{2}(\frac{(x_2-\hat{\mu}_2)}{\hat{\sigma}_2})^2}))^\theta)^{\frac{1}{\theta}})$$

In a next step, maximum likelihood estimation can be applied in order to find the MLE required. This is achieved by transforming the likelihood function using a log transformation and solving the maximization problem with respect to the parameters of interest θ.

$$L = \prod_{t=1}^{500} exp(-((-ln(\frac{1}{\sqrt{2\pi}\hat{\sigma}_1} e^{-\frac{1}{2}(\frac{(x_1-\hat{\mu}_1)}{\hat{\sigma}_1})^2}))^\theta + (-ln(\frac{1}{\sqrt{2\pi}\hat{\sigma}_2} e^{-\frac{1}{2}(\frac{(x_2-\hat{\mu}_2)}{\hat{\sigma}_2})^2}))^\theta)^{\frac{1}{\theta}})$$

The log transformation yields

$$logL = \sum_{t=1}^{500} (-((-ln(\frac{1}{\sqrt{2\pi}\hat{\sigma}_1} e^{-\frac{1}{2}(\frac{(x_1-\hat{\mu}_1)}{\hat{\sigma}_1})^2}))^\theta + (-ln(\frac{1}{\sqrt{2\pi}\hat{\sigma}_2} e^{-\frac{1}{2}(\frac{(x_2-\hat{\mu}_2)}{\hat{\sigma}_2})^2}))^\theta)^{\frac{1}{\theta}})$$

$$logL = \sum_{t=1}^{500} (-((-ln(\frac{1}{\sqrt{2\pi}\hat{\sigma}_1}) + \frac{1}{2}(\frac{(x_1-\hat{\mu}_1)}{\hat{\sigma}_1})^2))^\theta + (-ln(\frac{1}{\sqrt{2\pi}\hat{\sigma}_2} + \frac{1}{2}(\frac{(x_2-\hat{\mu}_2)}{\hat{\sigma}_2})^2))^\theta)^{\frac{1}{\theta}})$$

where $\theta \geq 1$.

In turn, the first order condition implies that the derivative with respect to θ must be equal to 0.

$$\frac{\partial logL}{\partial \theta} = 500(-\frac{1}{\theta}((-ln(\frac{1}{\sqrt{2\pi}\hat{\sigma}_1} e^{-\frac{1}{2}(\frac{(x_1-\hat{\mu}_1)}{\hat{\sigma}_1})^2}))^\theta + (-ln(\frac{1}{\sqrt{2\pi}\hat{\sigma}_2} e^{-\frac{1}{2}(\frac{(x_2-\hat{\mu}_2)}{\hat{\sigma}_2})^2}))^\theta)^{\frac{1}{\theta}-1})(...)$$
$$(...)(\theta(-ln(\frac{1}{\sqrt{2\pi}\hat{\sigma}_1} e^{-\frac{1}{2}(\frac{(x_1-\hat{\mu}_1)}{\hat{\sigma}_1})^2}))^{\theta-1} + \theta(-ln(\frac{1}{\sqrt{2\pi}\hat{\sigma}_2} e^{-\frac{1}{2}(\frac{(x_2-\hat{\mu}_2)}{\hat{\sigma}_2})^2}))^{\theta-1})) = 0$$

$$\frac{\partial logL}{\partial \theta} = 500(-\frac{1}{\theta}((-ln(\frac{1}{\sqrt{2\pi}\hat{\sigma}_1}) + \frac{1}{2}(\frac{(x_1-\hat{\mu}_1)}{\hat{\sigma}_1})^2))^\theta + (-ln(\frac{1}{\sqrt{2\pi}\hat{\sigma}_2} + \frac{1}{2}(\frac{(x_2-\hat{\mu}_2)}{\hat{\sigma}_2})^2))^\theta)^{\frac{1}{\theta}-1})(...)$$
$$(...)(\theta(-ln(\frac{1}{\sqrt{2\pi}\hat{\sigma}_1} + \frac{1}{2}(\frac{(x_1-\hat{\mu}_1)}{\hat{\sigma}_1})^2))^{\theta-1} + \theta(-ln(\frac{1}{\sqrt{2\pi}\hat{\sigma}_2} + \frac{1}{2}(\frac{(x_2-\hat{\mu}_2)}{\hat{\sigma}_2})^2))^{\theta-1} = 0$$

By solving for θ, the maximum likelihood estimator determined. In Python, drawing on the copulas library from Data to AI Lab at MIT, the maximum likelihood estimator $\hat{\theta}$ is calculated

$$\hat{\theta} = 1.5533$$

1.3 Model 4

$R_i = \mu_i + \sigma_i \epsilon_i$, where ϵ_i is standard t-distributed with ν_i degrees of freedom, for i = 1, 2. The parameters μ_i and σ_i , i=1,2, are the same as for model M2. The vector (ϵ_1, ϵ_2)' possesses a Gaussian copula with correlation parameter $\tilde{\rho}$.

In order to finally get an MLE for $\tilde{\rho}$, the first step is to plug in the standardized returns into the t-distribution with 499 degrees of freedom. This number results as there is 500 observations to be considered and 1 parameter to be estimated.

$$f_{t,1}(r_i, 499) = \frac{\Gamma(\frac{499+1}{2})}{\sqrt{499\pi}\Gamma(\frac{499}{2})}(1 + \frac{r_i}{499})^{-(\frac{499+1}{2})}$$

In the bivariate case, the random vector R is given by

$$f_{t,1}(r_1, r_2, 499, \rho) = \frac{\Gamma(\frac{499+2}{2})}{\Gamma(\frac{499}{2})\pi 499\sqrt{1-\rho^2}}(1 + \frac{r_1 - 2\rho r_1 r_2 + r_2^2}{499(1-\rho^2)})^{-(\frac{499+2}{2})}$$

where $\Gamma$is the gamma function, $\Gamma(z) = \int_0^\infty t^{z-1}e^{-t}dt$.

The Gaussian copula is defined by

$$C(u_1, u_2) = (\Phi_2(\Phi^{-1}(u_1)), \Phi^{-1}(u_2); \tilde{\rho})$$

where Φ^{-1} denotes the inverse cumulative distribution function of a standard normal and Φ_2 is the joint cumulative distribution function of a bivariate normal distribution with mean vector zero and covariance matrix equal to the correlation matrix $\sum$ as found under section 4.4.3.

Plugging in the t-distributed density function yields

$$C(u_1, u_2) = (\Phi_2(\Phi^{-1}(\frac{\Gamma(\frac{499+1}{2})}{\sqrt{499\pi}\Gamma(\frac{499}{2})}(1 + \frac{r_i}{499})^{-(\frac{499+1}{2})})), \Phi^{-1}(\frac{\Gamma(\frac{499+1}{2})}{\sqrt{499\pi}\Gamma(\frac{499}{2})}(1 + \frac{r_i}{499})^{-(\frac{499+1}{2})}); \tilde{\rho})$$

Formally, the likelihood function is defined in order to construct reasonable maximum likelihood estimators.

$$L = \prod_{t=1}^{500} (\Phi_2(\Phi^{-1}(\frac{\Gamma(\frac{499+1}{2})}{\sqrt{499\pi}\Gamma(\frac{499}{2})}(1 + \frac{r_i}{499})^{-(\frac{499+1}{2})})), \Phi^{-1}(\frac{\Gamma(\frac{499+1}{2})}{\sqrt{499\pi}\Gamma(\frac{499}{2})}(1 + \frac{r_i}{499})^{-(\frac{499+1}{2})}); \tilde{\rho}))$$

Applying the log transformation results in

$$logL = \prod_{t=1}^{500} ln(\Phi_2(\Phi^{-1}(\frac{\Gamma(\frac{499+1}{2})}{\sqrt{499\pi}\Gamma(\frac{499}{2})}(1 + \frac{r_i}{499})^{-(\frac{499+1}{2})})), \Phi^{-1}(\frac{\Gamma(\frac{499+1}{2})}{\sqrt{499\pi}\Gamma(\frac{499}{2})}(1 + \frac{r_i}{499})^{-(\frac{499+1}{2})}); \tilde{\rho}))$$

Setting $\frac{\partial logL}{\partial \rho} = 0$ and solving for ρ delivers the MLE $\hat{\rho}$. Alternatively, one can refer to the invariance principle and see that the estimates for $\hat{\mu}, \hat{\sigma}$ and $\hat{\rho}$

$$\hat{\sigma}_1 = 0.0062363$$

$$\hat{\sigma}_2 = 0.0064969$$

$$\hat{\rho} = 0.465021$$

2 Question II

Using model M1 for R, simulate the portfolio distribution (10,000 simulations of the portfolio return) and estimate 1-day portfolio's value-at-risk and expected shortfall at 90%, 95% and 99% confidence levels.

First, the expected shortfall is defined.

$$ES_{\alpha} = -E(R \mid R \leq -VaR_{\alpha})$$

Therefore, it describes the expected loss when the return actually is below the VaR. To get an estimate of the expected shortfall from an empirical return distribution, the following formula can be used:

$$ES_{\alpha} = \frac{-1}{\sum\limits_{t=1}^{T} \delta(R_t \leq -VaR_{\alpha} R_t)} \sum_{t=1}^{T} \delta(R_t \leq -VaR_{\alpha}) R_t,$$

where $\delta(q) = 1$ if q is true (return at t is below or equal to $-VaR_{\alpha}$) and zero otherwise. In other words, this expression yields the average R_t among those observations where $R_t \leq -VaR_{\alpha}$.
Value-at-risk is defined as the maximum loss that will be incurred on the portfolio with a given level of confidence over a specified holding period, based on the distribution of price changes over a given historical observation period. In mathematical terms, the value-at-risk is generally given by

$$VaR_{\alpha}(L) = -inf\left\{x \epsilon \mathbb{R} \mid \mathbb{P}\left[L \leq x\right] \geq 1 - \alpha\right\}$$

where $F_L$is the culmulative distribution function of L and L is a random variable that gives future profit and losses. This can be reformulated to get the idea more intuitively such that

$$Pr(R \leq -VaR_{1-\alpha}) = \alpha$$

This is under the assumption that the returns are normally distributed. The probability that a occuring return is smaller than the negative value-at-risk for a given confidence level $1 - \alpha$ is equal to α. However, in question ii, the risk measures have to be calculated from the empirical distribution as outlined in model 1. For this purpose, the values $\{x_i : i = 1, ..., n\}$ are sorted such that $x_{(1)} \leq x_{(2)} \leq ... \leq x_{(n)}$, then

$$VaR_{\alpha}(X) = -x_{(k^*)},$$

where

$$k^* = min\left\{k = 1, ..., n \mid \sum_{i=1}^{k-1} p_{(i)} < 1 - \alpha \leq \sum_{i=1}^{k} p_{(i)}\right\}.$$

After having defined the risk measures, respective results are obtained for the different confidence levels. In order to conduct the simulation, 10'000 random numbers were generated between 0 and 500. Then, 10'000 random returns from this sorted vector were drawn given the empirical cumulative distribution function. Drawing on this, the portfolio returns were calculated using the weights given in the assignment. As a result of these steps, one can find the respective, estimated values for both the expected shortfall and the value-at-risk assuming the empirical distribution of the data given.

	Expected shortfall	Value-at risk
90%	-0.0121	-0.0073
95%	-0.0139	-0.0107
99%	-0.0139	-0.0157

Table 1: Empirical Risk Mesures

After observing these results, one can discuss and distinguish the properties of the two risk measures. Compared to the variance, VaR provides a more sensible measure of the risk of the portfolio since it focuses on losses. However, the concept of value-at-risk has received some critique. While the value at risk is a useful risk measure, it has the property that it does not make a distinction between a loss that is just below the VaR level and a loss that is significantly below it. Therefore, VaR only describes whether the outcome is in the tail of the return distribution, but fails to state how far out. On the other hand, the expected shortfall is the expected loss when the return actually is below the VaR, while $VaR_\alpha$is the minimum loss that will happen with a $1 - \alpha$ probability. Moreover, value-at-risk generally fails to distinguish between the down-side of the return distribution (risk) and the upside (potential). The value-at-risk is one way of focusing on the downside. In addition, the VaR concept has poor aggregation properties. In particular, the VaR for a portfolio is not necessarily (weakly) lower than the portfolio of the VaRs, which contradicts the notion of diversification benefits.

3 Question III

In contrast to question II, where R was assumed to be empirically distributed, the expected shortfall is defined under the assumption that R follows a gaussian distribution (model 2) and that $R_i = \mu_i + \sigma_i \epsilon_i$, where ϵ_i is standard t-distributed with ν_i degrees of freedoms (model 4), respectively. Therefore the risk measures can be determined as follows:

The Expected shortfall assuming normal distribution of returns is

$$ES_\alpha = -\mu + \frac{\phi(c_{1-\alpha})}{1-\alpha}\sigma,$$

where $c_{1-\alpha}$ is the $1-\alpha$ quantile of $N(0,1)$ distribution. This can be calculated using the sample mean and standard deviation. For the VaR, if $L \sim N(\mu, \sigma^2)$ is normally distributed, then it holds

$$VaR_\alpha(L) = -\mu + \Phi^{-1}(\alpha)\sigma,$$

where Φ is the cumulative distribution function of the standard normal and Φ^{-1} is its quantile function. Furthemore, we set $z_\alpha = \varPhi^{-1}(\alpha)$.

3.1 Model 2

Model 2 has the following return distribution and risk measures. As you increase the confidence level, the expected shortfall and VaR increase. As shown in Figure 1, the expected shortfall also lies below the VaR. This corresponds to to the definition and intuition behind both risk measures. Since model 2 consists of abivariate gaussian, it might underestimate the probability of extreme returns. As a consequence, the absolute VaR might be estimated too optimistic (see Question V).

No. of Obs.	Exp. SF	VaR
Exp. Short Fall		
90%	-0.0096	-0.0077
95%	-0.0113	-0.0098
99%	-0.0148	-0.0137

Table 2: Risk Measures for Model 2

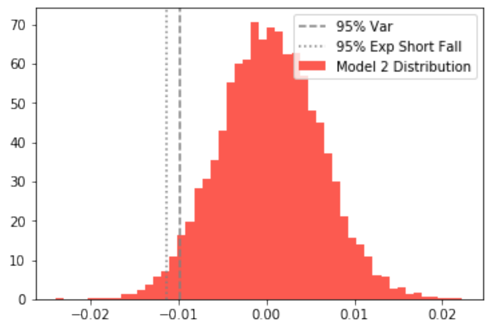


Figure 1: Model 2 Returns and VaR

3.2 Model 4

Model 4 has the following return distribution and risk measures. It exhibits larger values for the absolute expected shortfall and VaR than model 2. This is due to the fact that the simulated values exhibit a more leptokurtic distribution meaning the fat tails simulate a higher tail risk of the underlying assets. Since one of the stylized facts of asset returns states that stock returns exhibit fat tails, model 4 might actually be better suited for return simulation and more accurate risk measure estimation.

No. of Obs.	Exp. SF	VaR
Exp. Short Fall		
90%	-0.0125	-0.0101
95%	-0.0148	-0.0128
99%	-0.0192	-0.0179

Table 3: Risk Measures for Model 4

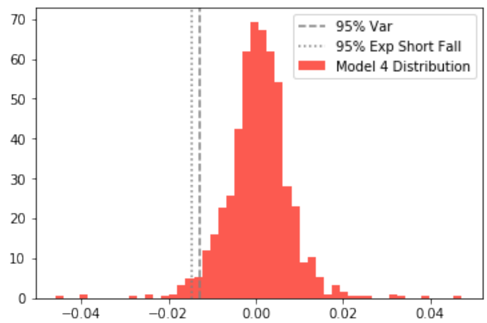


Figure 2: Model 4 Returns and VaR

4 Question IV

When calculating the expected shortfall and value at risk with different amounts of obseravtions *(100,200,500,1000)*, you can see that the risk measures deviate over time. This shows that volatility changed over the course of the last 1000 observations. Especially between the 100th last return and the 200th last return one can see an increase in risk. The same holds for the period between the 500th and 1000th return. Between the 200th and 500th one can observe a decrease in risk.

No. of Obs.	100	200	500	1000
Exp. Short Fall				
90%	-0.0103	-0.0130	-0.0094	-0.0131
95%	-0.0121	-0.0153	-0.0112	-0.0155
99%	-0.0156	-0.0198	-0.0146	-0.0201

Table 4: Expected shortfall for different number of hist. observations

No. of Obs.	100	200	500	1000
VaR				
90%	-0.0074	-0.0095	-0.0077	-0.0102
95%	-0.0095	-0.0122	-0.0098	-0.0130
99%	-0.0135	-0.0172	-0.0136	-0.0183

Table 5: Value at risk for different number of hist. observations

5 Question V

When simulating the model 2 with a rolling window of 200 observations, the VaR was violated 191 times (5.63%) This is displayed in Figure 3. A simulation of Model 4 however, yields 117 violations (3.45%) of the VaR (Figure 4). From this you can conclude that model 2 underestimates tail behaviour since the VaR is violated more often than the confidence level of 95%. The simulations of model 4 results in 1-day-ahead VARs which are violated far less often. This indicates that model 4 is suited better for risk management purposes in that regard.

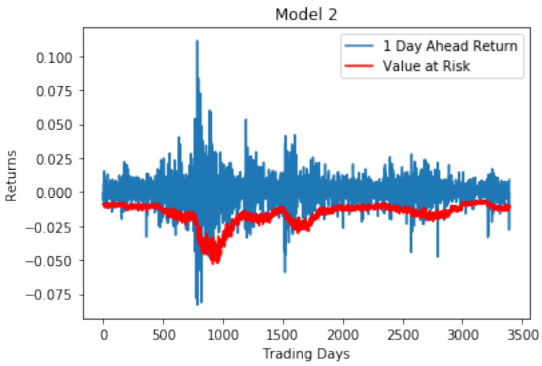


Figure 3: Model 2 Returns and VaR

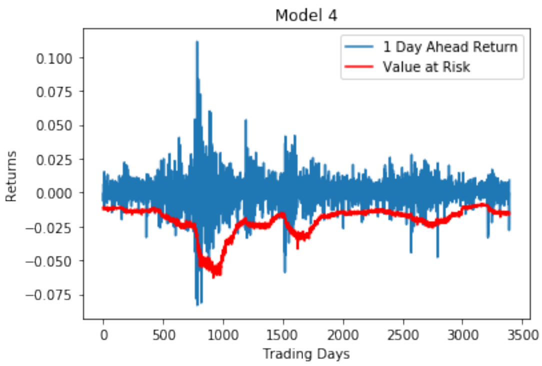


Figure 4: Model 4 Returns and VaR

When analyzing the means of all VaR estimates, one finds that model 2 yields a mean VaR of -0.0159, whereas model 4 gets a mean of -0.02058. Consequently the diffrence between the extreme losses and the VaR and smaller in the simulation of model 4. This shows that model 4 accounts better for tail risk.

Quantitative_Risk_Management

December 3, 2018

0.1 Quantitative Risk Management Assignement

Giuliano Cunti Lukas Schreiner Due: 2.Dec.2018

```
In [1]: #Import Libraries
        import numpy as np
        import pandas as pd

        from copulas.bivariate.base import Bivariate
        from copulas.multivariate.gaussian import GaussianMultivariate
        import scipy.stats as st

        import matplotlib.pyplot as plt

        #Set random number seed in order to generate coherent results
        np.random.seed(1)
```

```
In [2]: #Disable Warnings
        from warnings import filterwarnings
        filterwarnings('ignore')
```

```
In [3]: #Import Data MXEUC Index
        data1 = pd.read_excel('qrm18HSG_assignmentdata.xlsx', usecols = "A,B" ,
                              names = [ 'Date', 'MXEUC Index']).set_index('Date')
        data1 = data1.iloc[1:]
        #Import Data SPX Index
        data2 = pd.read_excel('qrm18HSG_assignmentdata.xlsx', usecols = "C,D" ,
                              names = [ 'Date', 'SPX Index']).set_index('Date')
        data2 = data2.iloc[1:]

        #Merge Data
        data = pd.concat([data1, data2], axis = 1, join_axes = [data1.index])

        #Convert to numerical values interpolate missing values
        data['SPX Index'] = pd.to_numeric(data['SPX Index'])
        data['MXEUC Index'] = pd.to_numeric(data['MXEUC Index'])
        data['SPX Index'] = data['SPX Index'].interpolate()
```

```
#Calculate Returns of the Indices
data['MXEUC Returns'] = data['MXEUC Index']/data['MXEUC Index'].shift(1) - 1
data['SPX Returns'] = data['SPX Index']/data['SPX Index'].shift(1) - 1

data['LT Returns'] = (0.3 * data['MXEUC Returns'] + 0.7 * data['SPX Returns'])

#Only take the latest 500 Oberservations (Questions i,ii, iii)
data_500 = data.iloc[-500:]

data_500.head()
```

Out[3]:

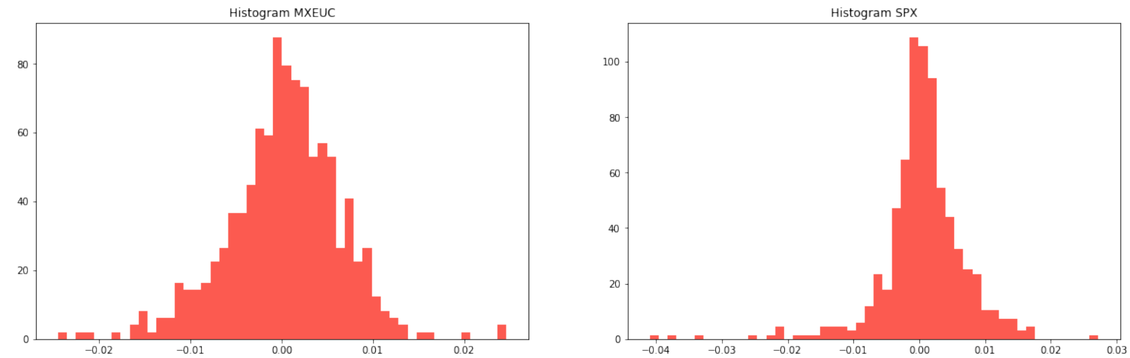
	MXEUC Index	SPX Index	MXEUC Returns	SPX Returns	LT Returns
Date					
2016-11-14	164.7258	2776.9236	0.002048	0.000029	0.000635
2016-11-15	165.4102	2798.3500	0.004155	0.007716	0.006648
2016-11-16	164.9615	2794.7027	-0.002713	-0.001303	-0.001726
2016-11-17	165.9808	2807.9370	0.006179	0.004735	0.005169
2016-11-18	165.3529	2801.6820	-0.003783	-0.002228	-0.002694

0.1.1 Question i)

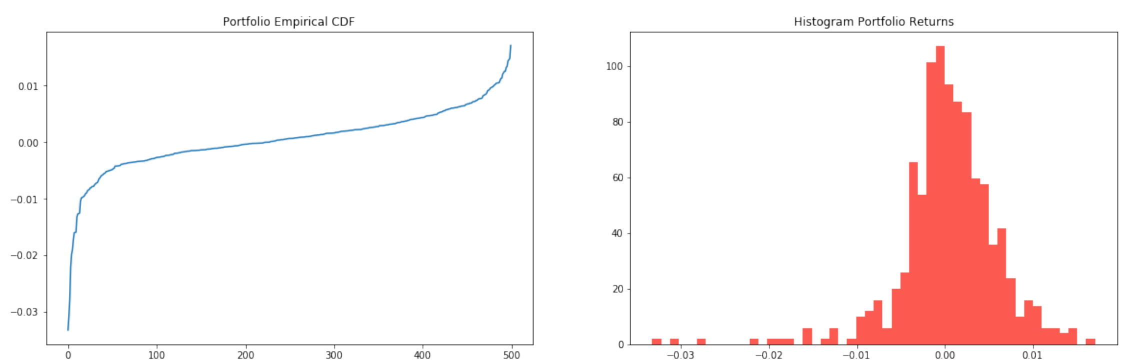
Model 1

In [4]:
```
#Calculate the emprical CDF
empricial_cdf = np.array([data_500.sort_values('MXEUC Returns')\
                          ['MXEUC Returns'].values,
                           data_500.sort_values('SPX Returns')\
                          ['SPX Returns'].values,
                           data_500.sort_values('LT Returns')\
                          ['LT Returns'].values])
```

In [5]:
```
#Plot CDF and histogramm of the MXEUC
figure_indices = plt.figure(figsize = (20,6))

chart_1  = figure_indices.add_subplot(121)
chart_1.hist(empricial_cdf[0], 50 ,density = True, facecolor = 'xkcd:coral')
chart_1.set_title('Histogram MXEUC')

chart_2  = figure_indices.add_subplot(122)
chart_2.hist(empricial_cdf[1], 50 ,density = True, facecolor = 'xkcd:coral')
chart_2.set_title('Histogram SPX')

plt.show()
```




```
In [6]: #Plot CDF and histogramm of the SPX
        figure_portfolio = plt.figure(figsize = (20,6))

        chart_1  = figure_portfolio.add_subplot(121)
        chart_1.plot(empricial_cdf[2])
        chart_1.set_title('Portfolio Empirical CDF')

        chart_2  = figure_portfolio.add_subplot(122)
        chart_2.hist(empricial_cdf[2], 50 ,density = True, facecolor = 'xkcd:coral')
        chart_2.set_title('Histogram Portfolio Returns')

        plt.show()
```



Model 2

```
In [7]: #Calculate Sample Means and Covariance Matrix
        def sample_moments_indices(data):

            means = [np.mean(data['MXEUC Returns']),
                     np.mean(data['SPX Returns'])]
```

```
    stds  = [np.std(data['MXEUC Returns']),
             np.std(data['SPX Returns'])]

    cov_mat = np.cov(data['MXEUC Returns'],
                     data['SPX Returns'], ddof = 0)

    return means, cov_mat, stds
```

```
In [38]: #Calculate the sample means of the latest 500 obervations
         means, cov_mat, stds =  sample_moments_indices(data_500)
         print('Means:', means)
         print('Covariance:', cov_mat)
         print('Std: ',stds)
         print('Corr: ', cov_mat[1][0]/(np.sqrt(cov_mat[0][0])*np.sqrt(cov_mat[1][1])))
```

```
Means: [0.00027679853531966693, 0.000588613647558554]
Covariance: [[3.88919478e-05 1.88413372e-05]
 [1.88413372e-05 4.22103025e-05]]
Std:  [0.006236340900598036, 0.006496945626430392]
Corr:  0.4650210831096565
```

Model 3

```
In [9]: #Normalize the data
        data_500['MXEUC Returns Normalized'] = (data_500['MXEUC Returns'] - means[0])/\
                                               (np.sqrt(cov_mat[0][0]))
        data_500['SPX Returns Normalized']   = (data_500['SPX Returns'] - means[1])/\
                                               (np.sqrt(cov_mat[1][1]))

        #Apply the normalized data on a CDF to get probablities
        data_500['MXEUC RN CDF'] = st.norm.cdf(data_500['MXEUC Returns Normalized'].values)
        data_500['SPX RN CDF']   = st.norm.cdf(data_500['SPX Returns Normalized'].values)
```

```
In [10]: #Create an array that contains only the return data
         copula_data = np.array([data_500['MXEUC Returns'], data_500['SPX Returns']]).T
```

```
In [11]: #Fit a bivariate gumbel copula
         #The copulas library from Data to AI Lab at MIT is used
         #https://pypi.org/project/copulas/

         gumbel = Bivariate(copula_type = 'Gumbel')
         gumbel.fit(copula_data)
         gumbel.compute_theta()
```

```
Out[11]: 1.5533515142571417
```

Model 4

```
In [12]: def model_4(data, size):

             #Calculate covariance matrix
             cov_mat  = np.corrcoef(data['MXEUC Returns'], data['SPX Returns'])

             #Calculate sample momemts
             means, _, stds = sample_moments_indices(data)

             #Degrees of freedom
             v = 5
             # Random chisquared RV W
             W = np.random.chisquare(df = v, size = size)
             # Define bivariate gaussian RV Z
             Z = np.random.multivariate_normal(means, cov_mat, size = size).T

             # Calculate X (see 4.4.3 on assignment sheet)
             # X is t-distributed with  degrees of freedom and correlation parameter .
             # Represents the error term in model 4
             X = np.sqrt(v/W)*Z

             #Calculate estimates for the return series
             #Ri = meani + sigmai * Xi
             MXEUC_estimate = means[0] + np.dot(stds[0],X[0])
             SPX_estimate   = means[1] + np.dot(stds[1],X[1])

             # Calculate portfolio return
             portfolio_return = np.add(np.dot(0.3,MXEUC_estimate),np.dot(0.7,SPX_estimate))

             return portfolio_return
```

0.1.2 Question ii)

```
In [13]: #Define Expetected Shortfall
         def exp_shortfall(alpha, data):

             mu = np.mean(data)
             sigma = np.std(data)

             return -(alpha**(- 1) * st.norm.pdf(st.norm.ppf(alpha)) * sigma - mu)

         #Define Value at Risk
         def var(alpha, data):
             return - (np.mean(data) - st.norm.ppf(alpha) * np.std(data))

         #Empirical Var with flexibel alpha level
         def emp_var(alpha, data):

             index = int(np.around(len(data) *alpha))
```

```
    returns_sorted = np.sort(data)

    return returns_sorted[index]

#Empirical Expected Shortfall
def emp_expshort(alpha, data):

    varalpha = var(alpha, data)

    deltas = []

    #Check wheter the indicator function delta is one or zero for all datapoints and
    for i in range(0,len(data)):
        deltas.append(int(data[i] < varalpha))

    ES = (-1/np.sum(deltas))*np.sum(deltas * data)

    return -ES
```

In [14]:
```
#Create table with risk measures
def calculate_risk_measures(data, name):

    #define dictionary to store the values in
    output = dict.fromkeys(['Exp. SF','VAR'])

    #Calculate the Expected Short Fall
    output['Exp. SF'] = [exp_shortfall(0.1, data),
                         exp_shortfall(0.05, data),
                         exp_shortfall(0.01, data)]

    #Calculate the Value at Risk
    output['VAR'] =  [var(0.1, data),
                      var(0.05, data),
                      var(0.01, data)]

    output_table = pd.DataFrame(output, index = ['90%', '95%', '99%'])
    output_table.index.name = name

    return output_table

#Create table with empirical risk measures
def calculate_emp_risk_measures(data, name):

    #define dictionary to store the values in
    output = dict.fromkeys(['Emp. Exp. SF','Emp. VAR'])

    #Calculate the Empirical Expected Short Fall
```

```
    output['Emp. Exp. SF'] = [emp_expshort(0.1, data),
                              emp_expshort(0.05, data),
                              emp_expshort(0.01, data)]

    #Calculate the Empirical Value at Risk
    output['Emp. VAR'] =  [emp_var(0.1, data),
                           emp_var(0.05, data),
                           emp_var(0.01, data)]

    output_table = pd.DataFrame(output, index = ['90%', '95%', '99%'])
    output_table.index.name = name

    return output_table
```

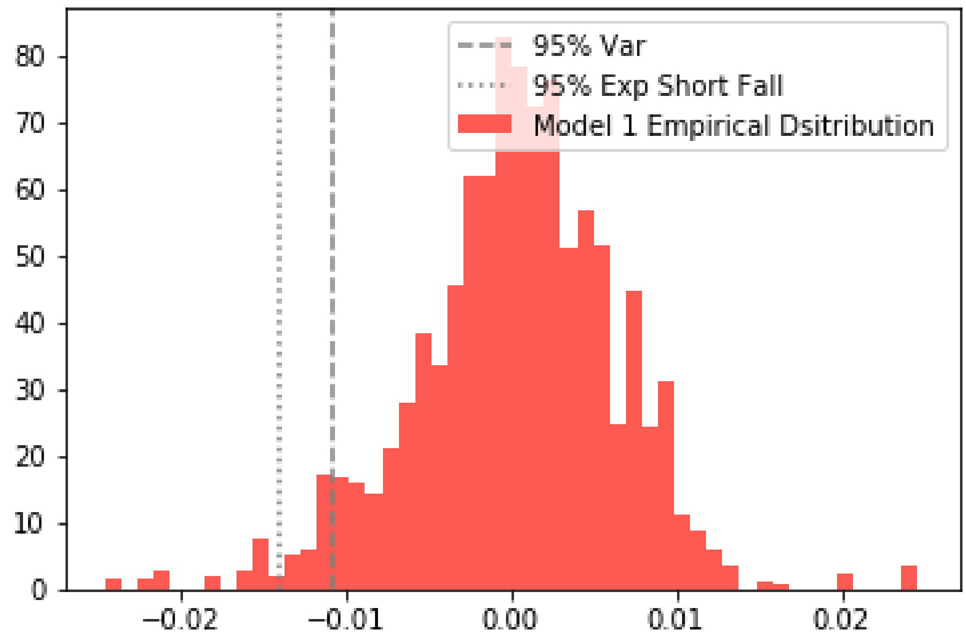
Model 1

In [15]:
```
#Generate 10'000 random numbers bewteen 0 and 500
rand_int = np.random.randint(0,500,10000)
#Draw 10'000 random returns from the sorted return verctor (empricial_cdf)
random_return_series_m1 = np.array([empricial_cdf[0][rand_int],
                                    empricial_cdf[1][rand_int]])
#Calculate Portfolio Return
port_ret_m1 = (0.3 * random_return_series_m1[0] + 0.7 * random_return_series_m1[0])
```

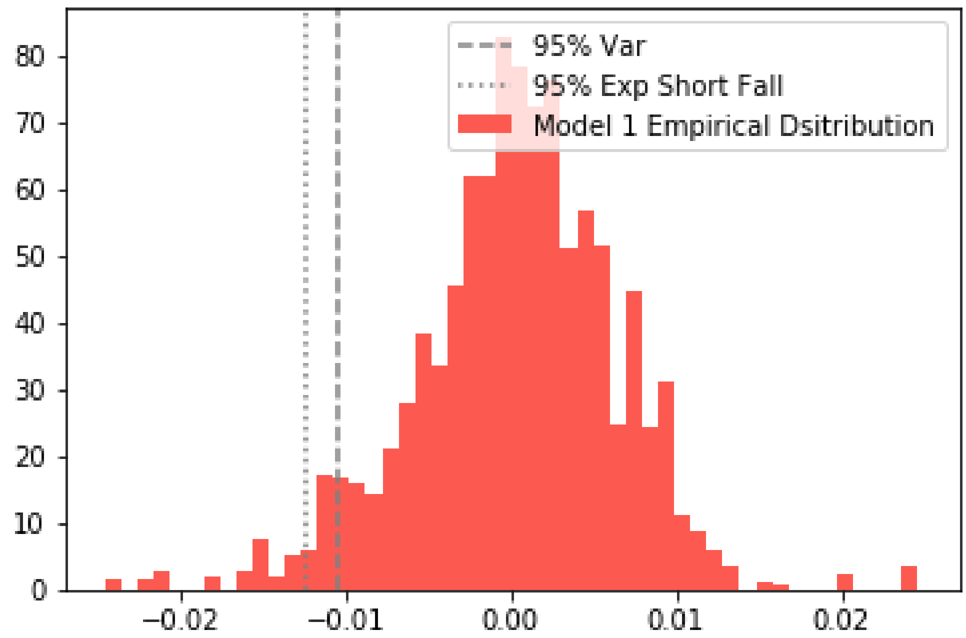
In [16]:
```
#Calculate Emprical Risk Measures for Model 1
rm_emp_model_1 = calculate_emp_risk_measures(port_ret_m1, 'Model 1')
rm_emp_model_1
```

Out[16]:
```
          Emp. Exp. SF  Emp. VAR
Model 1
90%          -0.012189 -0.007389
95%          -0.013946 -0.010744
99%          -0.017983 -0.015797
```

In [17]:
```
chart = plt.hist(port_ret_m1, 50,
                 density = True,
                 facecolor = 'xkcd:coral',
                 label = 'Model 1 Empirical Dsitribution')

chart = plt.axvline(x = rm_emp_model_1['Emp. VAR']['95%'], color = 'grey',
                    linestyle = '--',
                    label ='95% Var')

chart = plt.axvline(x = rm_emp_model_1['Emp. Exp. SF']['95%'], color = 'grey',
                    linestyle = ':',
                    label ='95% Exp Short Fall')
plt.legend()
plt.show()
```




```
In [18]: #Calculate Risk Measures for Model 1
         rm_model_1 = calculate_risk_measures(port_ret_m1, 'Model 1')
         rm_model_1

Out[18]:            Exp. SF       VAR
         Model 1
         90%     -0.010546 -0.008238
         95%     -0.012450 -0.010486
         99%     -0.016177 -0.014702

In [19]: chart = plt.hist(port_ret_m1, 50,
                          density = True,
                          facecolor = 'xkcd:coral',
                          label = 'Model 1 Empirical Dsitribution')

         chart = plt.axvline(x = rm_model_1['VAR']['95%'], color = 'grey',
                             linestyle = '--',
                             label ='95% Var')

         chart = plt.axvline(x = rm_model_1['Exp. SF']['95%'], color = 'grey',
                             linestyle = ':',
                             label ='95% Exp Short Fall')

         plt.legend()
         plt.show()
```



0.1.3 Question iii)

Model 2

In [20]:
```
#Model 2
#Create a random return series
random_return_series_m2 = np.random.multivariate_normal(means,
                                                         cov_mat,
                                                         size = (10000))

#Calculate Portfolio Return
port_ret_m2 = np.add(np.dot(0.3,[i[0] for i in random_return_series_m2]),
                     np.dot(0.7,[i[1] for i in random_return_series_m2]))
```

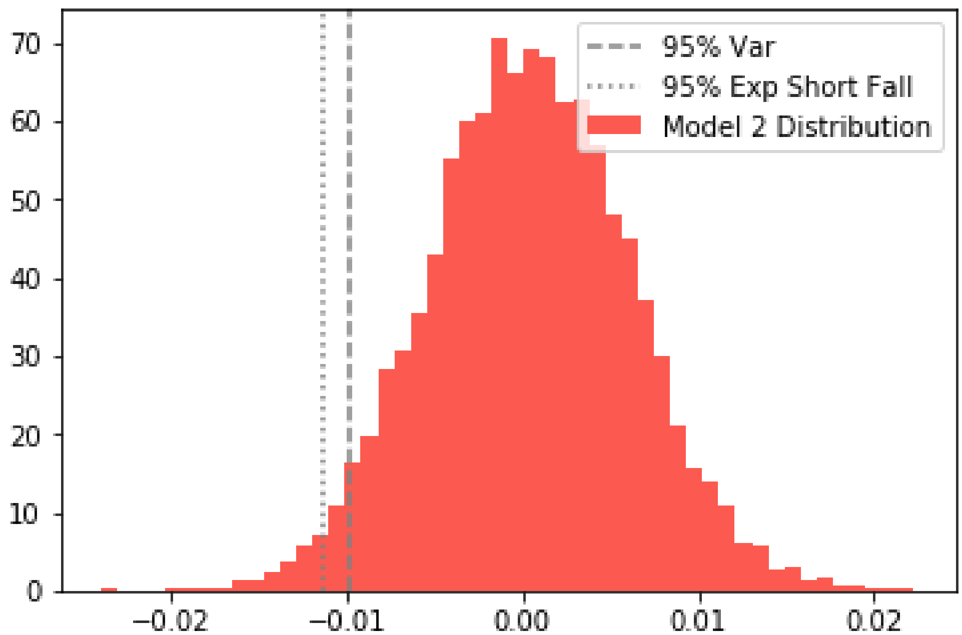
In [21]:
```
#Calculate Risk Measures for model 2
rm_model_2 = calculate_risk_measures(port_ret_m2, 'Model 2')
rm_model_2
```

Out[21]:

Model 2	Exp. SF	VAR
90%	-0.009593	-0.007706
95%	-0.011347	-0.009776
99%	-0.014779	-0.013658

In [22]:
```
chart = plt.hist(port_ret_m2, 50,
                 density = True,
```

```
                         facecolor = 'xkcd:coral',
                         label = 'Model 2 Distribution')

chart = plt.axvline(x = rm_model_2['VAR']['95%'], color = 'grey',
                    linestyle = '--',
                    label ='95% Var')

chart = plt.axvline(x = rm_model_2['Exp. SF']['95%'], color = 'grey',
                    linestyle = ':',
                    label ='95% Exp Short Fall')
plt.legend()
plt.show()
```



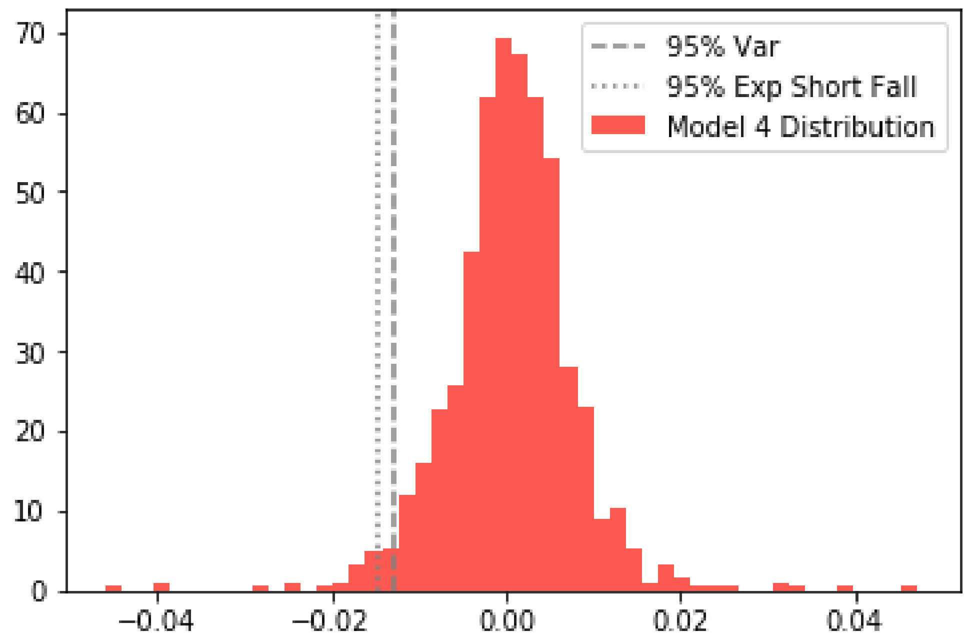
Model 4

```
In [23]: random_return_series_m4 = model_4(data_500, 1000)
         rm_model_4 = calculate_risk_measures(random_return_series_m4, 'Model 4')
         rm_model_4
```

```
Out[23]:            Exp. SF       VAR
         Model 4
         90%      -0.012466 -0.010105
         95%      -0.014753 -0.012806
         99%      -0.019232 -0.017871
```

```
In [24]: chart = plt.hist(random_return_series_m4, 50,
                          density = True,
```

```
                     facecolor = 'xkcd:coral',
                     label = 'Model 4 Distribution')

chart = plt.axvline(x = rm_model_4['VAR']['95%'], color = 'grey',
                    linestyle = '--',
                    label ='95% Var')

chart = plt.axvline(x = rm_model_4['Exp. SF']['95%'], color = 'grey',
                    linestyle = ':',
                    label ='95% Exp Short Fall')
plt.legend()
plt.show()
```



0.1.4 Question iv)

In [25]:
```
def historic_EF(data):
    #Data has to be a pandas series with 1000 historical returns

    #Define dictionary to store the values in
    output_EF = dict.fromkeys(['100','200','500','1000'])

    #Calculate the Expected Short Fall for the latest 100 observations
    output_EF['100'] = [exp_shortfall(0.1,    data.iloc[-100:]),
                        exp_shortfall(0.05,   data.iloc[-100:]),
                        exp_shortfall(0.01,   data.iloc[-100:])]
```

```
    #Calculate the Expected Short Fall for the latest 200 observations
    output_EF['200'] =  [exp_shortfall(0.1,  data.iloc[-200:]),
                         exp_shortfall(0.05, data.iloc[-200:]),
                         exp_shortfall(0.01, data.iloc[-200:])]

    #Calculate the Expected Short Fall for the latest 500 observations
    output_EF['500'] =  [exp_shortfall(0.1,  data.iloc[-500:]),
                         exp_shortfall(0.05, data.iloc[-500:]),
                         exp_shortfall(0.01, data.iloc[-500:])]

    #Calculate the Expected Short Fall for the latest 1000 observations
    output_EF['1000'] =  [exp_shortfall(0.1,  data.iloc[-1000:]),
                          exp_shortfall(0.05, data.iloc[-1000:]),
                          exp_shortfall(0.01, data.iloc[-1000:])]

    output_table_EF = pd.DataFrame(output_EF, index = ['90%', '95%', '99%'])
    output_table_EF.index.name = 'EF Hist. R'

    return output_table_EF

def historic_VAR(data):
    #Data has to be a pandas series with 1000 historical returns

    #Define dictionary to store the values in
    output_VAR = dict.fromkeys(['100','200','500','1000'])

    #Calculate the VAR for the latest 100 observations
    output_VAR['100'] = [var(0.1,    data.iloc[-100:]),
                         var(0.05,   data.iloc[-100:]),
                         var(0.01,   data.iloc[-100:])]

    #Calculate the VAR for the latest 200 observations
    output_VAR['200'] =  [var(0.1,  data.iloc[-200:]),
                          var(0.05, data.iloc[-200:]),
                          var(0.01, data.iloc[-200:])]

    #Calculate the VAR for the latest 500 observations
    output_VAR['500'] =  [var(0.1,  data.iloc[-500:]),
                          var(0.05, data.iloc[-500:]),
                          var(0.01, data.iloc[-500:])]

    #Calculate the VAR for the latest 1000 observations
    output_VAR['1000'] =  [var(0.1,  data.iloc[-1000:]),
                           var(0.05, data.iloc[-1000:]),
                           var(0.01, data.iloc[-1000:])]

    output_VAR = pd.DataFrame(output_VAR, index = ['90%', '95%', '99%'])
    output_VAR.index.name = 'VAR Hist. R'
```

```
        return output_VAR
```

In [26]:
```
#Historic expected shortfall for the MXEUC with diffrent number of observations
hist_EF = historic_EF(data['LT Returns'])#
hist_EF
```

Out[26]:
```
                  100       200       500      1000
EF Hist. R
90%         -0.010307 -0.013025 -0.009448 -0.013105
95%         -0.012103 -0.015310 -0.011191 -0.015471
99%         -0.015619 -0.019784 -0.014604 -0.020101
```

In [27]:
```
#Historic expected shortfall for the SPX with diffrent number of observations
historic_VAR(data['LT Returns'])
```

Out[27]:
```
                   100       200       500      1000
VAR Hist. R
90%          -0.007413 -0.009520 -0.007756 -0.010232
95%          -0.009533 -0.012218 -0.009814 -0.013024
99%          -0.013510 -0.017277 -0.013675 -0.018262
```

0.1.5 Question v)

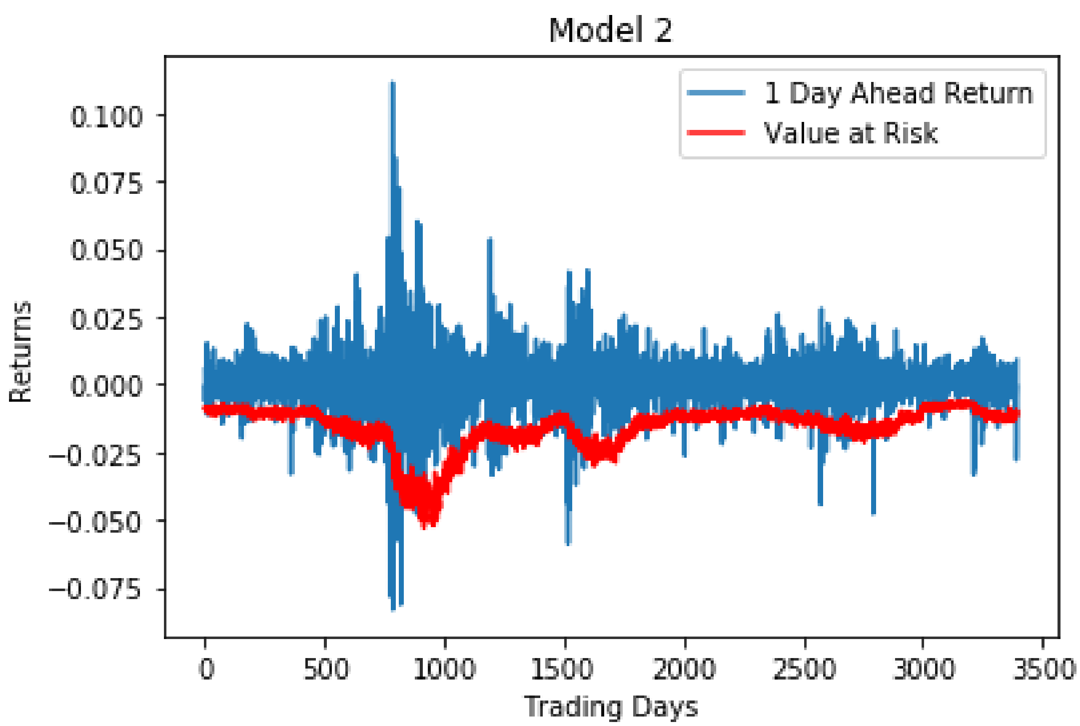
Model 2

In [40]:
```
#Initilaized List to store the values for var and the 1 day ahead portfolio return
rw_var = []
day_ahead_return = []

for i in range(1,len(data.index) - 201):
    #Get the rolling window
    rw_data = data.iloc[i:i + 200]

    #Save the actual 1 day ahead (of the window) return in the list
    day_ahead_return.append(data['LT Returns'].iloc[i + 200 + 1])

    #Calculate the moments for that window
    rw_means, rw_cov_mat, _ = sample_moments_indices(rw_data)

    #Simulated index returns on the basis of the rolling window moments
    rw_sim_returns = np.random.multivariate_normal(rw_means, rw_cov_mat, size = 100)

    #Calculte the simululated portfolio return
    #'[i[0] for i in rw_sim_returns]' gives the first/second elements
    #from the 2 dimensional vector 'rw_sim_returns'
    rw_port_returns = np.add(np.dot(0.3,[i[0] for i in rw_sim_returns]),
                             np.dot(0.7,[i[1] for i in rw_sim_returns]))
```

```
        #Calculate Var and store it in the list
        rw_var.append(var(0.05, rw_port_returns))

    print(np.mean(rw_var))
```

```
-0.015934407171025063
```

```
In [29]: plt.plot(day_ahead_return, label = '1 Day Ahead Return')
         plt.plot(rw_var, label = 'Value at Risk', color = 'red')

         plt.xlabel('Trading Days')
         plt.ylabel('Returns')
         plt.legend()
         plt.title('Model 2')
         plt.show()
```




```
In [30]: #Count the number of times the the VAR was violated
         count = 0
         loc = []
         for i in range(0,len(rw_var)):
             if rw_var[i] > day_ahead_return[i]:
                 count += 1
                 loc.append(i)
```

```
print('Model 2: The Var was violated ',count,' times which is ',
      round(count/len(rw_var)*100,2),' Percent')
```

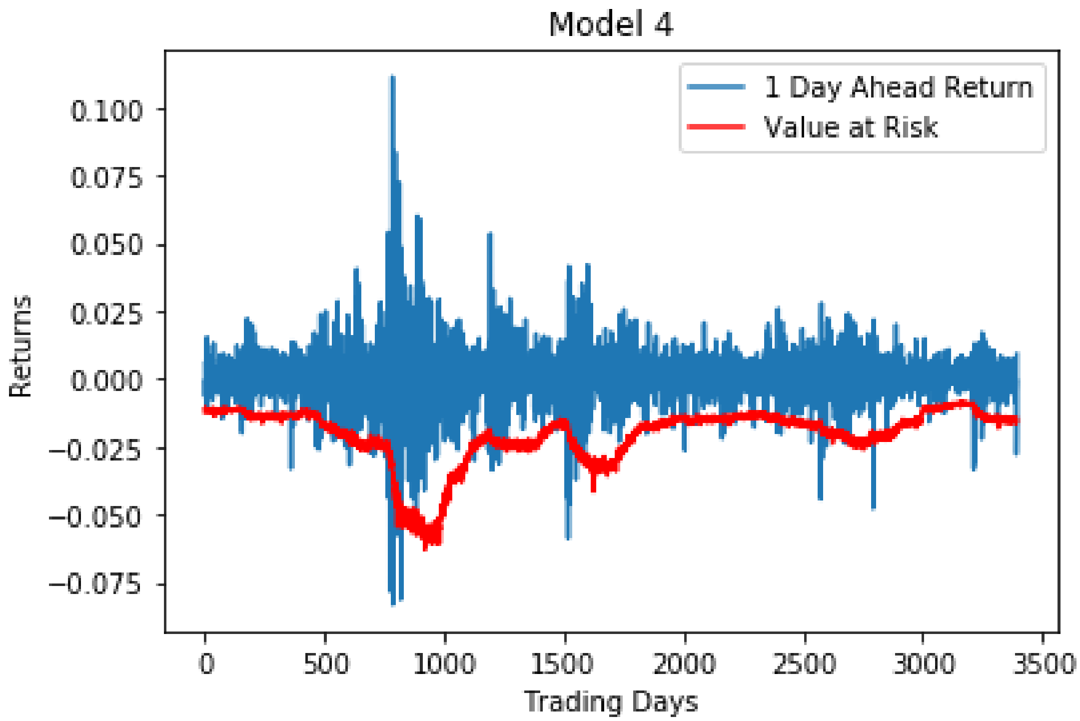
```
Model 2: The Var was violated  191  times which is  5.63  Percent
```

Model 4

```
In [41]: rw_var = []
         day_ahead_return = []

         for i in range(1,len(data.index) - 201):

             #Get the rolling window
             rw_data = data.iloc[i:i + 200]

             #Save the actual 1 day ahead (of the window) return in the list
             day_ahead_return.append(data['LT Returns'].iloc[i + 200 + 1])

             #Calculate the moments for that window
             rw_means, rw_cov_mat, rw_stds = sample_moments_indices(rw_data)

             #Simulated index returns on the basis of the rolling window moments
             rw_port_returns = model_4(rw_data, size = 1000)

             #Calculate Var and store it in the list
             rw_var.append(var(0.05, rw_port_returns))

         print(np.mean(rw_var))
```

```
-0.020586392775727 97
```

```
In [32]: plt.plot(day_ahead_return, label = '1 Day Ahead Return')
         plt.plot(rw_var, label = 'Value at Risk', color = 'red')

         plt.xlabel('Trading Days')
         plt.ylabel('Returns')
         plt.title('Model 4')
         plt.legend()
         plt.show()
```




```
In [33]: #Count the number of times the the VAR was violated
         count = 0
         loc = []
         for i in range(0,len(rw_var)):
             if rw_var[i] > day_ahead_return[i]:
                 count += 1
                 loc.append(i)

         print('Model 4: The Var was violated ',count,' times which is ',
               round(count/len(rw_var)*100,2),' Percent')
```

```
Model 4: The Var was violated  117  times which is  3.45  Percent
```