MATH GR5320

Term Project Report

Risk Calculation System

Group 5 Members:

Yunxiao Zhao, yz3380

Wanzhen Jiang, wj2289

Instructor:

Professor Harvey J. Stein

Dec 18th, 2019

Contents

1 Executive Summary

This documentation is a review of risk calculation system developed by Yunxiao Zhao and Wanzhen Jiang. The system takes a portfolio of stock and option positions, then calibrates both to historical data and takes the parameters as input. The main calculation includes: Parametric, Historical, and Monte Carlo VaR, as well as Historical and Monte Carlo ES. The system also includes the backtest results of the computed VaR against history.

The risk calculation system is adequate for the target risk assessment, as it is capable of taking arbitrary portfolios and parameters as inputs. Another desirable characteristic is convenience: users can acquire stock information and calculation results without downloading data manually. Because of restricted access to the Bloomberg terminal, it is not as convenient for users to add new options as they can do with stocks, which is a minor weakness we have detected.

2 Introduction

Due to the prevailing focus on risk management, a well-developed risk calculation system becomes critical in decision making. The review of this risk calculation system documents methodologies of calculation for each approach mentioned in the section above. In addition, it evaluates the model performance and justifies the suitability of model for our risk calculation.

3 Model Descriptions

In this risk calculation system, we are using three different methods to calculate the Value at Risk for a portfolio: Parametric VaR, Historical VaR and, Monte Carlo VaR. We are also using Historical ES and Monte Carlo ES to measure the Expected Shortfall for the portfolio. Each of these methods are calculated based on different assumptions. To begin with, we are going to explain VaR and ES and their properties. Then we will focus on discussing the model assumptions, model inputs, advantages and disadvantages for each method used to

evaluate VaR and ES.

3.1 Value at Risk (VaR) and Expected Shortfall (ES)

Value at risk (VaR) and Expected shortfall (ES) are two important risk measures used to quantify the level of financial risk associated with a portfolio.

Value at Risk VaR represents the maximum expected loss with a certain confidence level and within a given period of time. For example, the p level of VaR is X means that the loss will be less than or equal to X over p percent of the time. If the value of portfolio is V, then

$$\text{VaR}(V, T, p) = G^{-1}(p) \tag{1}$$

where

$$G(X) = P\left[V_0 - V_T \leq X\right] = E^P\left[1_{V_0 - V_T \leq X}\right] \tag{2}$$

VaR is elicitable, which means it can be defined as the minimizer of a suitable expected scoring function:

$$\rho(V) = \underset{X}{\text{argmin}}\, E[L(X, V)] \tag{3}$$

where $L(X, V)$ is the error being made when using X as an estimate of $\rho(V)$ and $E[L(X, V)]$ is the average error.

Another desirable property of VaR is robustness, given that it is insensitive to changes in the portfolio, changes in the historical data and changes in parameters.

A common criticism of VaR is that it is not coherent. Diversification can further increase the risk. Therefore, it does not satisfy the sub-additivity condition, which requires

$$\rho(V + V') \leq \rho(V) + \rho(V') \tag{4}$$

Expected Shortfall ES represents the expected loss condition on when it is greater than the value of the VaR calculated with that confidence level:

$$\mathrm{ES}(V, T, p) = -E^p \left[V_T - V_0 | V_T - V_0 < -\mathrm{VaR}(V_T, p) \right] \tag{5}$$

ES is a coherent measure of risk as diversification can never increase risk. However, it is harder to backtest ES than VaR given that ES is testable but not elicitable. In addition, it is not clear how actionable ES is, meaning that the value may tell us little what to do.

3.2 Parametric VaR

3.2.1 Model Assumption

The parametric method is the most common risk management technique for calculating the VaR. For a single stock, Parametric VaR makes simplifying assumption that stock price follows the Geometric Brownian Motion. For an option, we also assume the price of the underlying stock follows the Geometric Brownian Motion. When it comes to the calculation of portfolio, it is generally assumed that the portfolio either follows the GBM or it is normally distributed. In the mathematical description section, we are going to discuss two methods in calculating the Parametric VaR:

1. Assuming the whole portfolio follows GBM, we obtain the Parametric VaR by computing its drift and volatility.

2. It is assumed that each underlying stock follows possibly different but correlated GBM. When combining them all into one portfolio, the whole portfolio follows normal distribution. In this case, we would take into account the correlation coefficient between stocks to calculate the mean and standard deviation of the portfolio.

3.2.2 Model Input

The parametric method is simple because the only variables we need as inputs are the mean, standard deviation of the portfolio and the time horizon. In order to calculate the mean and standard deviation, we first need to calculate the mean returns and the covariance matrix for the portfolio.

3.2.3 Mathematical Description

Method 1: GBM assumption To obtain the drift and volatility of Geometric Brownian Motion, let S_i denotes the stock closing price at the end of i^{th} trading period and dt denotes the length of time interval between two consecutive trading periods expressed in year. Let u_i denote the logarithm of the daily return on the stock over the short time interval dt:

$$u_i = \ln\left(\frac{S_i}{S_{i-1}}\right), \qquad for\ i = 1, 2, 3 \cdots n \tag{6}$$

Then the unbiased estimator $\bar{u}$ of the logarithm of the returns is given by:

$$\bar{u} = \frac{1}{n}\sum_{i=1}^{n} u_i \tag{7}$$

The estimator of the standard deviation is given by:

$$v = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(u_i - \bar{u})^2} \tag{8}$$

It follows that:

$$\sigma = \frac{v}{\sqrt{dt}} \tag{9}$$

We can further obtain the expected annual rate of return or the drift μ:

$$\bar{u} = \left(\mu - \frac{1}{2}\sigma^2\right) dt \tag{10}$$

$$\mu = \frac{\bar{u}}{dt} + \frac{1}{2}\sigma^2 \tag{11}$$

GBM VaR can be calculated using the formula below by plugging in the drift and volatility parameters:

$$\text{VaR}(S, T, p) = S_0 - S_0 \text{e}^{\sigma\sqrt{T}\Phi^{-1}(1-p)+(\mu-\frac{\sigma^2}{2})T} \tag{12}$$

Method 2: Normal assumption All the portfolios in this risk calculation system are self-financing, meaning that variations of the portfolio are only due to variations of asset prices and not from an injection or withdrawal of money. In mathematical description,

$$V(t) = \sum_{i-1}^{n} h_i(t) S_i(t) \tag{13}$$

where

$h_i(t)$ is the number of shares of stock number 'i' in the portfolio at time t

$S_i(t)$ is the price of stock number 'i' in a frictionless market with trading in continuous time.

The portfolio ($h_1(t)$,. . . ,$h_n(t)$) is self-financing if

$$dV(t) = \sum_{i-1}^{n} h_i(t) dS_i(t) \tag{14}$$

Consider a portfolio of two assets, let V_t denotes the portfolio value at time t; let a and b denotes the number of shares for Stock 1 and Stock 2 respectively; let $S_{i,t}$ denotes the stock price at time t and ρ represents the correlation coefficient between the two stocks.

$$V_t = aS_{1,t} + bS_{2,t} \tag{15}$$

$$\text{d}S_i = \mu_i S_i \text{d}t + \sigma_i S_i \text{d}W_i \tag{16}$$

$$\text{d}W_1 \text{d}W_2 = \rho \text{d}t \tag{17}$$

$$E\left[S_{1,t} S_{2,t}\right] = S_{1,0} S_{2,0} \text{e}^{(\mu_1+\mu_2+\rho\sigma_1\sigma_2)t} \tag{18}$$

$$E\left[V_t\right] = aS_{1,0}\text{e}^{\mu_1 t} + bS_{2,0}\text{e}^{\mu_2 t} \tag{19}$$

$$E\left[V_t^2\right] = a^2 S_{1,0}^2 \mathrm{e}^{(2\mu_1+\sigma_1^2)t} + b^2 S_{2,0}^2 \mathrm{e}^{(2\mu_2+\sigma_2^2)t} + 2abS_{1,0}S_{2,0}\mathrm{e}^{(\mu_1+\mu_2+\rho\sigma_1\sigma_2)t} \tag{20}$$

$$\mathrm{var}\left[V_t\right] = E\left[V_t^2\right] - E\left[V_t\right]^2 \tag{21}$$

$$sd\left[V_t\right] = \sqrt{var\left[V_t\right]} \tag{22}$$

$$VaR\left[V\right] = V_0 - \left(E\left[V_t\right] - \phi^{-1}(1-p)sd\left[V_t\right]\right) \tag{23}$$

The equations above can be expanded and generalized if the portfolio consists of more than two stocks.

Windowed calibration vs. Exponentially weighted calibration When calculating Parametric VaR, the parameters can be calibrated either by the window length or exponential weighting. A windowed estimate with N points is the same as weighting scheme where the first N points are given a weight of 1/N and the remaining N points are given a weight of 0. To find the exponential weighting λ that best approximates a window size of N, we need to minimize the $L2$ norm of weights differences, which is

$$R = \sum_{i=0}^{N-1}\left(\frac{1}{N} - (1-\lambda)\lambda^i\right)^2 + \sum_{i=N}^{\infty}\left((1-\lambda)\lambda^i\right)^2 \tag{24}$$

$$= \frac{2\lambda^{N+1} + 2\lambda^N - \lambda(N+1) + N - 1}{(\lambda+1)N} \tag{25}$$

To minimize, we solve it numerically for the derivative being zero.

$$\frac{dR}{d\lambda} = \frac{2(2\lambda^{N+1} + \lambda^{N+2} + \lambda^N - \lambda)}{\lambda(\lambda+1)^2} \tag{26}$$

3.2.4 Advantages & Limitations

The advantage of parametric method is that it is simple and the computation time is minimal. Since Parametric VaR assumes the portfolio follows the Geometric Brownian Motion or even Normal distribution, the infeasibility of these assumptions in real life becomes one of its greatest limitation. In addition, Parametric VaR does not cope well with securities that have a non-linear payoff like options or mortgage-backed securities. It also underestimates VaR at

high confidence levels and overestimates VaR at low confidence levels.

3.3 Historical VaR

3.3.1 Model Assumption

Historical VaR is a better methodology to use if the distribution of the return series is undetermined. While Historical VaR does not make any assumptions about the distribution of historical changes, it generally assumes risk factors follow actual historical distribution. i.e. it assumes the future distribution of market changes equals to that in the past. There are two methods in calculating Historical VaR: 1) Absolute changes 2) Relative changes.

We are indifferent between absolute and relative changes when historical and current values are both low or both high. When historical values are low and current values are high, applying absolute changes would yield tiny changes while relative changes would yield huge changes. In contrast, under the circumstances of high historical and low current values, applying absolute changes could cause huge changes and relative changes may bring tiny changes.

If we believe a risk factor has constant absolute volatility (ABM), then absolute changes would be more representative. In practice, it is better to use absolute changes when dealing with spreads and use relative changes for rates and stock prices. If we believe a risk factor has constant relative volatility (GBM), we would use relative changes.

3.3.2 Model Input

In the calculation of Historical VaR, the historical data of the portfolio and the horizon period are needed as inputs.

3.3.3 Mathematical Description

1. Determine an appropriate time span of the VaR i.e 5-day VaR, 10-day VaR first.

2. Based on the time span chosen, calculate a series of changes for each underlying asset in the portfolio within the time span, as well as shares for each stock in the portfolio.

3. Multiply the losses of each stock by the number of shares to obtain a series of losses

4. Rank the series of losses from the lowest to the highest, and the portfolio loss with rank $n(1-\alpha)\%$ is the $(1-\alpha)\%$ VaR of the portfolio.

3.3.4 Advantages & Limitations

Similar to Parametric VaR, Historical VaR can be quickly computed. It does not assume normal distribution and can be easily applied to different periods of time. Historical VaR has a great limitation: it simply assumes the future will equal to the past, which is unrealistic in many ways. In addition, for stocks with little or none historical data, Historical VaR would be inapplicable.

3.4 Monte Carlo VaR

3.4.1 Model Assumption

Monte Carlo VaR is a much more complex analytical tool than parametric and historical methods. It simulates the risk factors and use the pricers to directly compute the VaR. While Monte Carlo VaR allows for an infinite number of possible scenarios, we are also exposing our system to huge model risks in determining the likelihood of any given path. Monte Carlo VaR generally assumes the stock price follows the Geometric Brownian Motion. In Monte Carlo VaR, options are valued using Black-Scholes model, which needs the following assumptions:

1. The risk-free rate r and volatility of the underlying asset σ are known and constant.

2. The underlying stock does not pay dividends during the option's life (We use the zero dividend Black-Scholes model).

3. The underlying asset price follows Geometric Brownian Motion.

4. All securities are perfectly divisible, meaning that it is possible to buy any fraction of a share.

5. Markets are efficient, which suggests that market movements cannot be predicted.

6. There are no transaction costs or taxes.

7. Short selling of securities with use of proceeds is permitted.

8. There are no risk-less arbitrage opportunities.

9. There is no jump in the underlying asset price

3.4.2 Model Input

In order to calculate Monte Carlo VaR, portfolio returns and stock returns, correlation matrix between stocks, the window size and the horizon are needed. For options, underlying price, interest rate, implied volatility, option type, maturity and strike price are needed.

3.4.3 Mathematical Description

The following steps are needed for the calculation:

- Obtain the drift and volatility first for each underlying stock in the portfolio.
- Generate a series of correlated random variables using the parameters in the first step.
- Use the formula below to generate the stock and option price for the portfolio.

For stocks:
Let t denotes the time horizon:

$$S_t = S_0 * \mathrm{e}^{\left(\mu - \frac{\sigma^2}{2}\right)t + \sigma W_t} \tag{27}$$

For options:
Black-Scholes model should be used to determine the theoretical value for a European call or

a European put option based on variables such as volatility, type of option, underlying stock price, time, strike price, and risk free rate.

$$c = S_0 N(d_1) - K\mathrm{e}^{-rT} N(d_2) \tag{28}$$

$$p = K\mathrm{e}^{-rT} N(-d_2) - S_0 N(-d_1) \tag{29}$$

where

$$d_1 = \frac{\ln(S_0/K) + (r + \sigma^2/2)T}{\sigma\sqrt{T}} \tag{30}$$

$$d_1 = \frac{\ln(S_0/K) + (r - \sigma^2/2)T}{\sigma\sqrt{T}} = d_1 - \sigma\sqrt{T} \tag{31}$$

where

c = Call option price

p = Put option price

S = Current stock price

K = Strike price

r = Risk free interest rate

T = Time to maturity

N refers to CDF of Normal distribution

- Sum up each asset multiplied by its number of shares to calculate the portfolio value.
- Rank the series of losses of the portfolio from the lowest to highest, which is calculated by subtracting V_t from V_0.
- The portfolio loss with rank $n(1-\alpha)\%$ is the $(1-\alpha)\%$ VaR of the portfolio.

3.4.4 Advantages & Limitations

There are many advantages for Monte Carlo VaR. It can model instruments with non-linear and path dependent payoff functions. It is also not affected as much by extreme events. Finally, it can use any statistical distribution to simulate the returns as far as comfortably possible.

The disadvantage is also obvious: the simulation is time consuming and complicated. As a result, it is costly to develop a Monte Carlo VaR engine. Similar to the drawbacks of parametric VaR, Monte Carlo VaR makes assumptions on the distribution of underlying stocks and the portfolio. The distribution might be infeasible in reality, which could lead the inaccurate simulation results.

3.5 Historical ES

3.5.1 Model Assumption

Same as historical VaR, historical ES assumes the future performance of stocks, options and portfolios is consistent with that in the past.

3.5.2 Model Input

Historical ES takes the historical data of the portfolio and the time horizon as inputs.

3.5.3 Mathematical Description

In order to get the value of Historical ES, we need:

1. First determine an appropriate time span of the ES.
2. Based on the time span chosen, calculate a series of changes for each underlying asset in the portfolio within the time span, as well as shares for each stock in the portfolio.
3. Multiplying the losses of each stock by the number of shares to obtain a series of losses.
4. Rank the series of losses from the highest to the lowest, and the historical ES at α significance level is the average of the worst losses $n\alpha\%$ in the tail.

3.5.4 Advantages & Limitations

Historical ES bears the same pros & cons as historical VaR: the simplicity of the approach can lead to fast and easy risk calculation, as it does not involve the calculation of drift and volatility. However, given that the future performance of stock will not always duplicate the past, historical ES could be unreliable. Since historical ES highly depends on the historical data, an effective risk analysis cannot be conducted if little or even no past data is provided.

3.6 Monte Carlo ES

3.6.1 Model Assumption

Same as Monte Carlo VaR, the Monte Carlo ES assumes that stocks follow the Geometric Brownian Motion and option price can be calculated using Black-Scholes model.

3.6.2 Model Input

Portfolio and stock returns, covariance matrix, and the time horizon are needed as inputs.

3.6.3 Mathematical Description

The mathematical description for Monte Carlo ES is similar to that in 3.4.3. Except in the last step, the Monte Carlo ES at a significance level is the average of the worst losses $n\alpha\%$ in the tail.

3.6.4 Advantages & Limitations

The pros & cons are the same for Monte Carlo VaR and ES.

4 Software Documentation

The risk calculation system is developed by applying knowledge introduced in the GR5320 Financial Risk Management and Regulations class through lectures and homework assignments. It is built and compiled in Python 3.7 and on Windows 10, while the user interface is written using Tkinter, a standard Python interface to the Tk GUI toolkit. The system aims to provide a user-friendly tool in evaluating portfolio risk measures under different scenarios based on users' choices. There are three main objectives in the system: VaR calculation and ES calculation as the market risk measures, and backtesting as the validation of VaR. We follow two major rules in designing the software: simplicity and versatility.

The complete software with all codes and documentation can be found at our GitHub repository: `https://github.com/yz3380/5320Project`.

4.1 Software Architecture

The system consists of four parts:

1. The "Option_Data" folder contains selected implied volatility data for options;
2. The "portfolio.txt" file stores the input portfolio positions;
3. The "my_utils.py" script stores all the utility functions for the complete calculation process;
4. The "RMSystem.py" script is the launcher implements GUI and initiates the program. The calculations use the aforementioned model formulas and assumptions on an arbitrary combination of stocks and a certain group of options.

4.1.1 Software Input

To make the operation of our system as simple as possible, we separate the inputs into three parts:

1. **Portfolio positions**: This is a text file located in the working directory of software. Users can set an arbitrary target portfolio consisting of stocks and available options.

2. **Built-in parameters**: These parameters can be manually adjusted when running the program. They are:

    - Investment starting time
    - Investment ending time
    - Portfolio value (for normalization purpose)
    - Level of probability for VaR
    - Level of probability for ES
    - Window size for parameter calibrations
    - Horizon period
    - Calibration method (window-sized or exponentially weighted)

3. **Historical data**: They are required only when the portfolio contains options. Due to restrictions of using Bloomberg API, we only have a limited number of implied volatility data for options as illustrations. They are downloaded from Bloomberg terminal and stored as daily values from Jan 1st, 2005 to Nov 22th, 2019. And the daily interest rate data is from the Federal Reserve website, with the same time period as volatility.

4.1.2 Software Assumption

Apart from the assumptions described in the model section, there are several explicit assumptions for the implementation of functions, including:

1. There are exactly 252 trading days each year, and the available dates for historical data are consistent for every stock and option. In reality, this assumption generally does not hold. For each year's historical data, one needs to calibrate it according to holidays and abnormal events.

2. The portfolio value does not change sign during the investment period. It is obvious that the total value cannot go negative for a pure long portfolio in practice. However, the loss for short positions are unbounded, which may lead to negative portfolio values and cause trouble in calibrating GBM parameters. In fact, the system still provides results when the values crosses through the point of zero, but with a warning message to notice the user that results shown in plots may be highly unreliable or completely trashed.

3. Short-term interest rate can be approximated by 1-year Fed rate. We scale the rate to 3 months and 6 months period and assume it remains constant in option pricing.

4. The historical data span is long enough for large window size and horizon period. Although users can set any investment period they want, the system will automatically raise an error message and stop the calibration if this requirement is not satisfied.

5. The options in the portfolio have everlasting properties, which means its time to maturity remains unchanged during the whole investment period.

6. All the options are at the money (ATM) options due to limited available data from Bloomberg.

4.1.3 Software Structure

There are two pages in the software user interface: the index page and the calculation page. The index page will provide general information about the software, including the author information, the current version of the software, and a help link to the software user guide on GitHub. The calculation page integrates all the functional buttons. It allows users to set parameters and customize their own experiments.

All the algorithms used in the system are rigorously designed in order to maximize the calculation efficiency. The number of loops utilized is minimized only for the tickers of assets, while all the time period based calculations are completed through Numpy vectorization. One exception is the higher-order running time (number of assets * length of the time period) on

the Monte Carlo approach, as we need a large number of paths (10000 as default) to ensure the stability of simulation results. Generally, as long as the number of assets in the portfolio is not too large, the whole calculation process will finish within 2 minutes (may vary for different CPUs), which is acceptable in practice.

See Appendix for a complete list of functions in the system.

4.1.4 Software Implementation

The implementation is straightforward. Once the calculation button is clicked, the system will check all the inputs first, raising an error if any input is inappropriate. Then two main functions, 'Read_Main' and 'Calculate_Main', will be implemented accordingly. Each main function will call a number of auxiliary functions to finish the computation task. Calculation results will be stored as Pandas data frame for the use of visualization.

Long/ short position The long/ short position of a portfolio is determined by the portfolio value at the initial investment period. If the portfolio has a positive value at the beginning, the system will treat it as a long portfolio, and vice versa.

Parameter calibration There are two methods for parameter calibration: one is window sized and the other is equivalent exponentially weighted. For window sized method, the system simply takes fixed length of portfolio return data and apply the GBM parameter estimation mentioned in previous section. For exponentially weighted method, the system will first approximate λ by optimizing L_2 distance between exponential weights and equal weights for a given window size. Then it will apply the calculated weights according to λ in estimating GBM parameters.

VaR calculation There are three types of VaR measurements, each containing two approaches. The parametric VaR is approximated by using either GBM or Normal assumptions. The historical VaR is using either relative or absolute change. And Monte Carlo VaR are either treated as one asset (MC One) or simulated by each underlying asset(MC Multi). For

MC Multi VaR, the system simulates each day's underlying prices by generating multivariate normal (*numpy.random.multivariate_normal*) based on daily correlation matrix in 10000 paths. Option prices are calculated using BlackScholes according to the underlying. If there are a large number of underlying stocks, the MC method will take longer time.

ES calculation The calculation approaches are similar to those of VaR.

Backtest The system performs a backtesting using the number of exceptions within a period of time. Exceptions stand for the situations where the actual loss in a portfolio violates the calculated VaR at that date. For instance, on date i, the VaR on that date is VaR(i). The actual observed 5-day loss would be Price(i) - Price($i + 5$). By comparing a 252-day actual loss to VaR, the number of exceptions within one year is obtained.

4.1.5 Software Output

The output consists of plots and datasets. Users can choose to visualize any of the following: the portfolio information, the VaR analysis, the ES analysis, and validation results. Each visualization can be implemented through a corresponding button on the control panel. For simplicity, the detailed calculation results are not displayed on the user interface. Instead, users can save the results into a CSV file by hitting the 'Save' button and keep the numbers for future reference.

4.1.6 Numerical Methodology

The parametric model in the system does not involve any numerical methodologies such as PDE grid solver of option pricing. However, when choosing VaR for historical and Monte Carlo simulation, we use 'higher' or 'lower' method in *np.quantile*, which means it takes the nearest neighbor with higher or lower value, to ensure the elicitable property of VaR. This methodology applied throughout the system is consistent with our previous model description.

4.2 User Guide

This section is the official user guide for MATH GR 5320 Term Project: Risk Calculation System developed by Yunxiao Zhao and Ophelia Jiang.

4.2.1 Installation

The system is developed in Python 3.7 on Windows 10. All packages used are included in the Python 3.7 Standard Library. If not, one may use pip commands to install the required packages. For example, in a *cmd* window try:

- `>python m pip install pandas`
- `>python m pip install pandas_datareader`
- `>python m pip install scipy`

Or in a JupyterNotebook try:

- `!pip install pandas`
- `!pip install pandas_datareader`
- `!pip install scipy`

For more installation help please visit the official Python document: `https://docs.python.org/3/installing/index.html`.

For Windows users, no additional actions are needed. The system will run properly as long as all the required packages are installed.
For Mac users, since the software is built using Tkinter GUI toolkit, if someone is using macOS 10.6 or later, the Apple-supplied Tcl/Tk 8.5 has serious bugs that can cause application crashes. To launch the software, DO NOT use the Apple-supplied Pythons. Instead, install and use a newer version of Python from python.org or a third-party distributor that supplies or links with a newer version of Tcl/Tk. See `https://www.python.org/download/mac/tcltk/#built-in-8-6-8` for more information.

4.2.2 Getting Started

Before opening the software, make sure the **'Option_Data' folder**, the **'Portfolio.txt'** file and the **'my_utils.py'** file are located under the **same working directory** with the software launchers. Once the setup is done, there are three versions of the launcher, one in Python script and two in JupyterNotebook.

To run the UI version in Python scripts (RCSystem.py):

1. Open a *cmd* window
2. Set the current working directory to where the launcher locates
3. Launch the software by
   >`python RCSystem.py`

To run the UI version in notebooks (RCSystem_GUI_Ver):

1. Open RCSystem_GUI_Ver.ipynb file in JupyterNotebook
2. Run the cell

To run the plain dash version in notebooks (RCSystem_Notebook_Ver):
This version is used for demonstration only when the previous two versions are unavailable due to unresolvable issues, for example, the incompetence under macOS 10.6. All the instructions in the following sections will focus on how to operate the system with the intentionally designed UI.

1. Open RCSystem_Notebook_Ver.ipynb file in JupyterNotebook
2. Select 'Run all cells' in JupyterNotebook toolbar

Once the software is launched, click the start button and switch to the calculation page. Figure 4.1 is how the index page looks like when the software is launched.

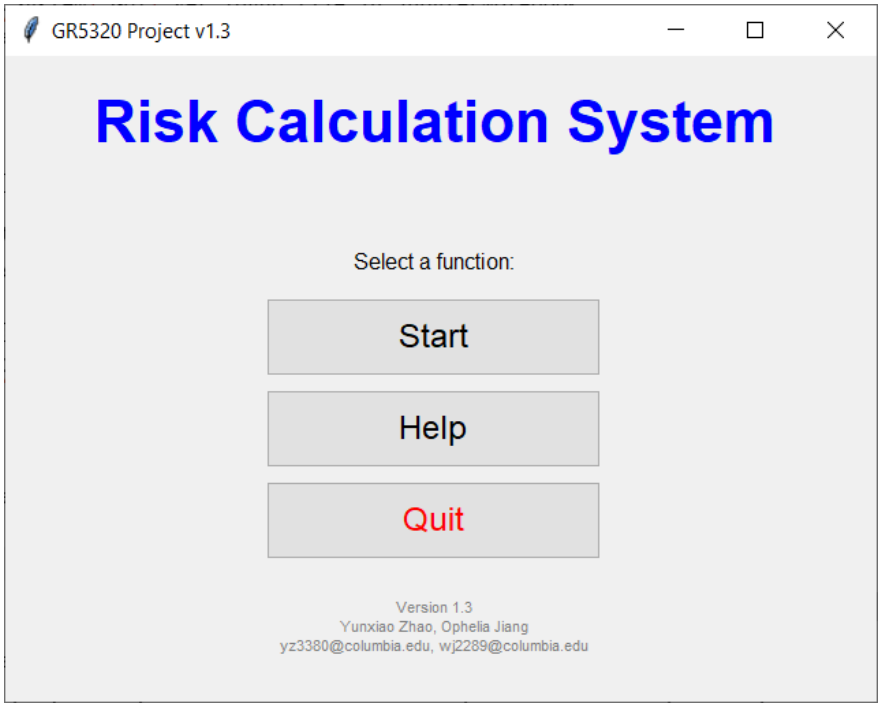

Figure 4.1: Index Page

4.2.3 Setup Portfolio

Before starting the calculation, one may want to create his own portfolio. The portfolio's positions are stored in the 'Portfolio.txt' file. Any saved changes to this file will affect the calculation in the software. If positions are modified after a calculation, redo calculation to update the results.

The stocks should be formatted as $< ticker >, < position >$, one in each line. The ticker should be on file on Yahoo finance and position should be in the form of floats, while negative numbers stand for short positions. For example:

AAPL, -3.5

The options should be formatted as $< underlying >, < position >, < type >, < maturity >$, one in each line. Due to limited access to Bloomberg terminal, we only have ATM options with 11 underlying as illustrations, they are AAPL, AMZN, BKNG, COF, CVS, DATE, GE, KO, NKE, NVDA, SPX, XRX. The position should be floats, with negative numbers stand

for short positions. The type should be either call or put. And maturity (in months) should be chosen from 3, 6 and 12 according to Bloomberg implied volatility data. For example:

GE, 5, put, 12

Once an option is chosen, the investment period will be restricted from Jan 1, 2015 to Nov 22, 2019 due to limited historical data of implied volatility and interest rate.

4.2.4 Change Parameters

There are eight parameters in total on the calculation page. See figure 4.2.

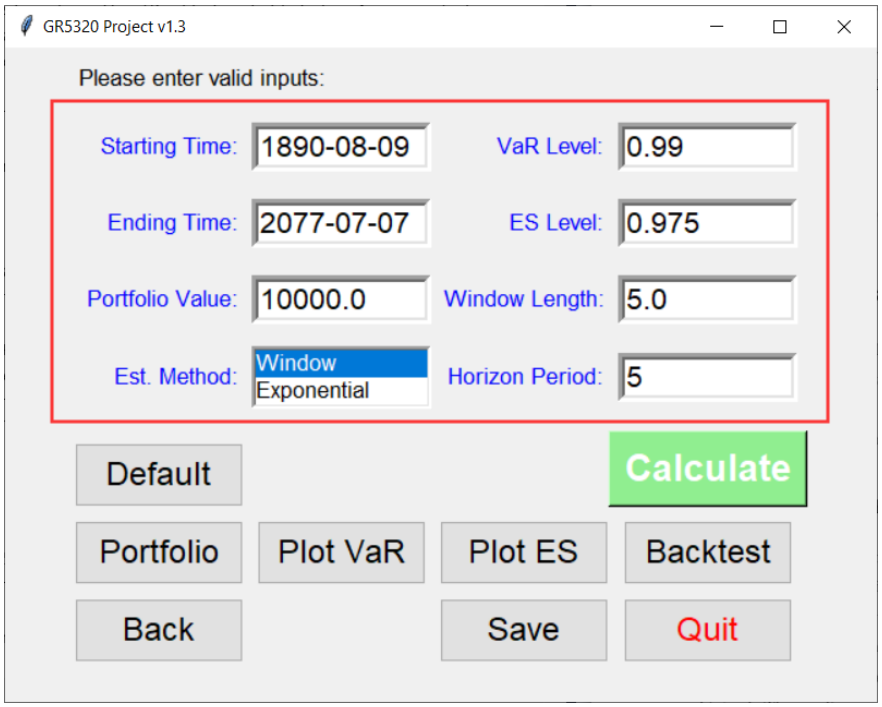

Figure 4.2: Parameter Entries

Rather than use the default parameters, one can customize the experiment by entering his own parameters:

1. **Starting Time:** String formatted as YYYY-MM-DD, the starting time of investment period

2. **Ending Time:** String formatted as YYYY-MM-DD, the ending time of investment period

3. **Portfolio Value:** Float > 0, the normalized portfolio value for VaR calculation

4. **Estimation Method:** Either "Window" or "Exponential" in calibrating GBM parameters

5. **VaR Level:** Float in (0, 1), the level of probability for VaR

6. **ES Level:** Float in (0, 1), the level of probability for ES

7. **Window Length:** Float > 0, number of years used for parameter calibration

8. **Horizon period:** Integer > 0, number of days used in defining VaR

The 'Default' button will set all the parameters to the default value.

4.2.5 Calculation

Click the 'Calculate' button if all parameters and inputs are set. If any of the requirements is not met, there will be a pop-up message informing the user to modify the inputs or parameters. Otherwise, the system will proceed to calculation and provide a notice when finishing. If the number of assets in the portfolio is not too large, it will take less than 2 minutes to complete the calculation process (may vary for different CPUs).

Notice that the time period will be automatically adjusted to fit for stocks with the shortest available historical data during the calculation.

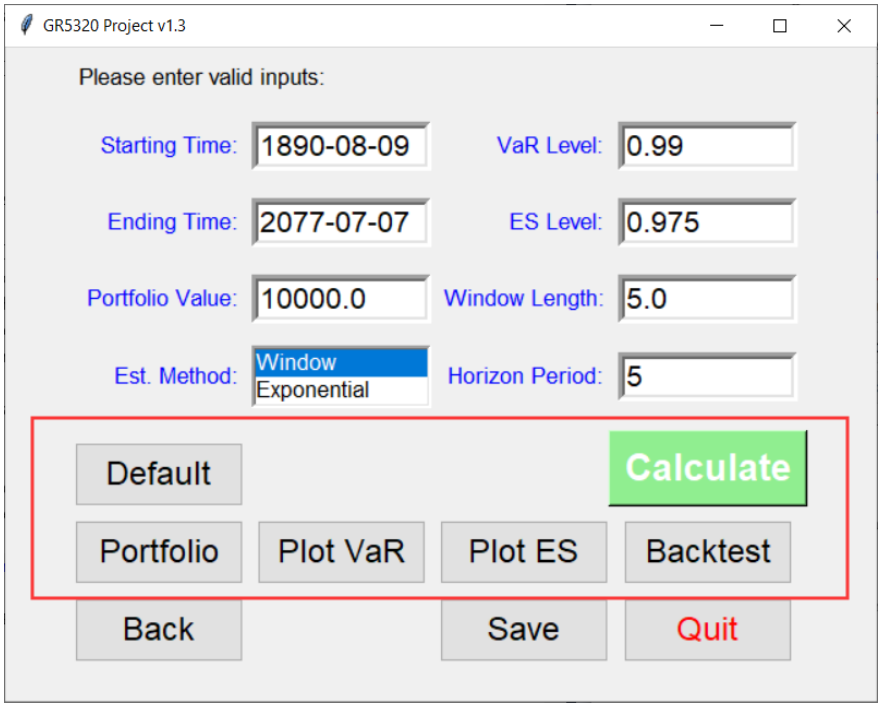

Figure 4.3: Functional Buttons

4.2.6 Plots

Once the calculation is done. All the analysis of results can be visualized.

To view the portfolio information, click the 'Portfolio' button. There will be 4 subplots: the initial assets allocation, the historical prices of all stocks, the value of a portfolio without normalization, and calibrated GBM parameters. A sample plot is shown in figure 4.4.

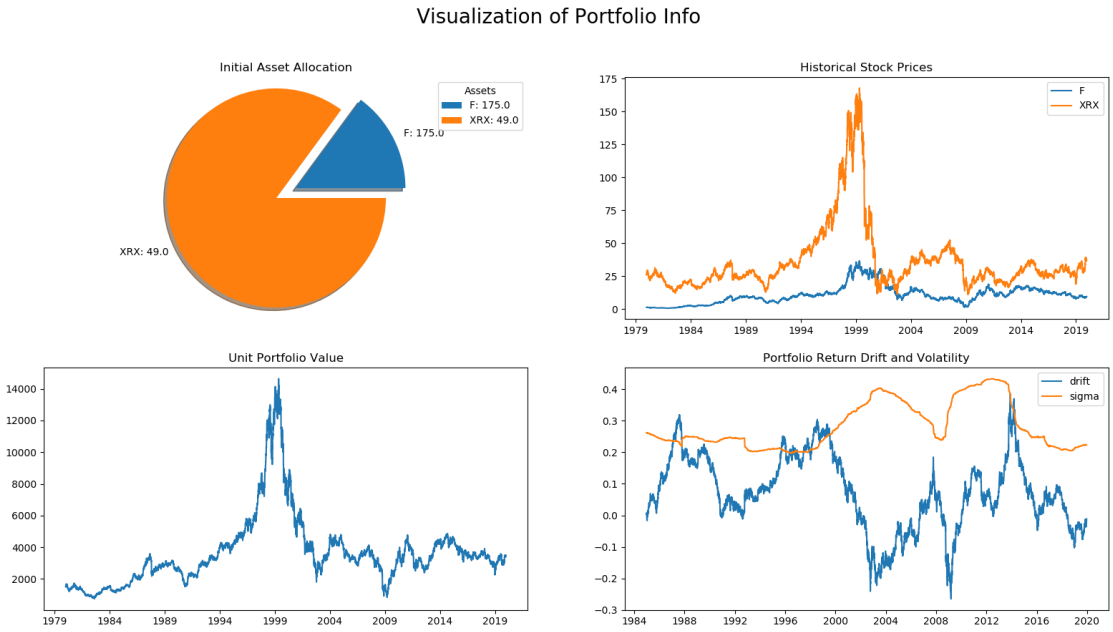

Figure 4.4: Portfolio Info

To view the analysis of VaR, click the 'VaR' button. There will be 4 subplots: the plot of Parametric VaR, the plot of Historical VaR, the plot of Monte Carlo VaR, and the plot of a comparison between all methods. See figure 4.5.

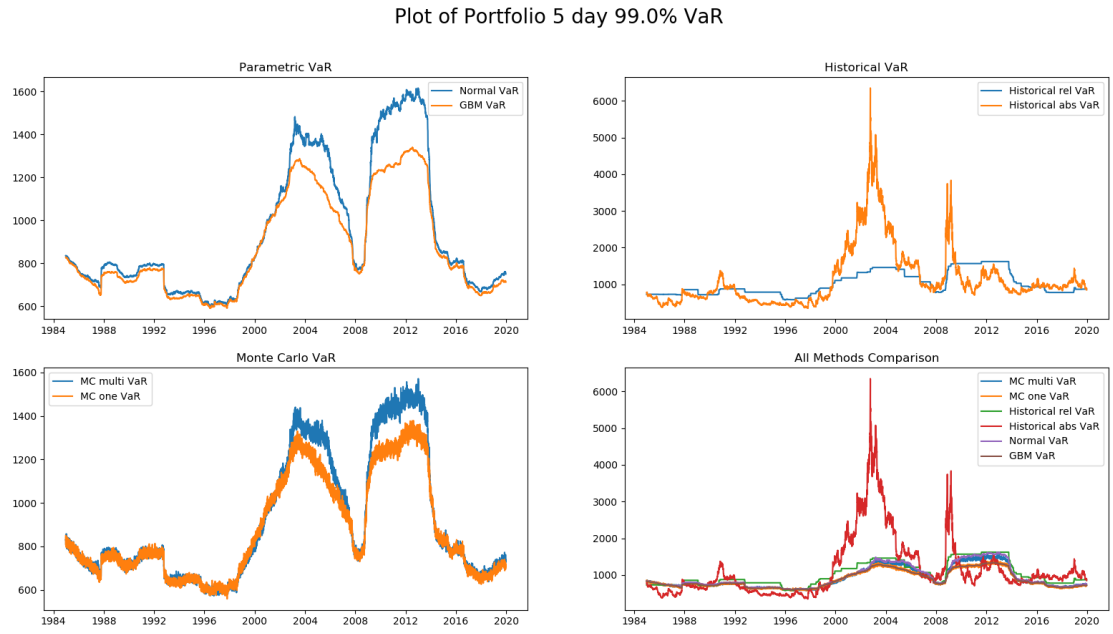

Figure 4.5: Analysis of VaR

To view the analysis of ES, click the 'ES' button. There will be 3 subplots: the plot of Historical ES, the plot of Monte Carlo ES, and the plot of a comparison between all methods. See figure 4.6.

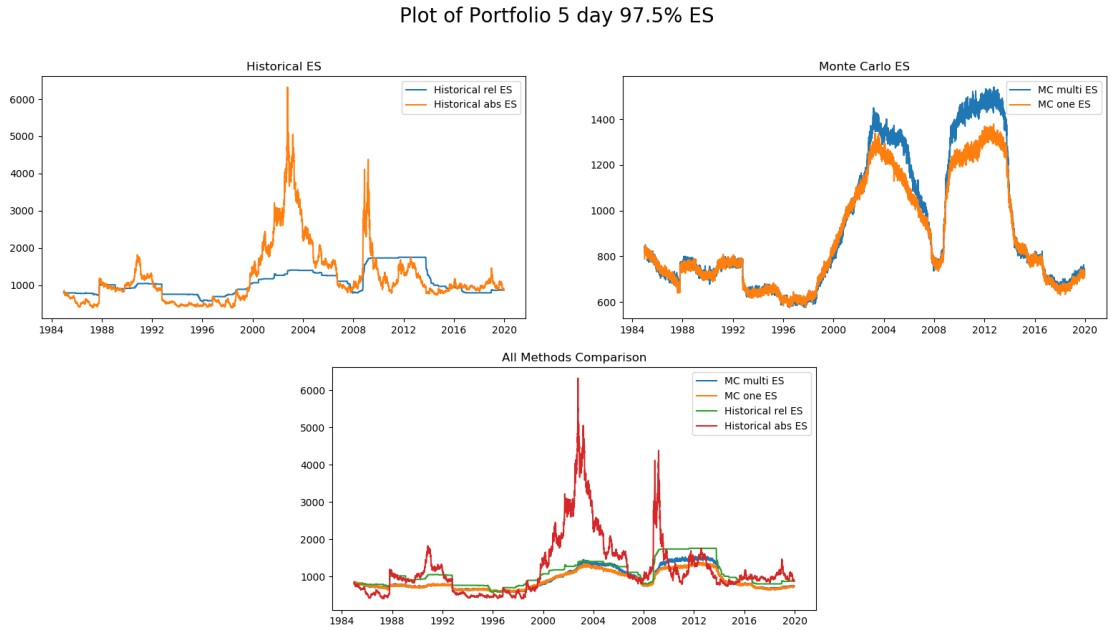

Figure 4.6: Analysis of ES

To view the analysis of validations, click the 'Backtest' button. There will be 2 subplots: one is calculated VaR vs. actual loss plot and the other one is the plot of the number of exceptions. As shown in figure 4.7.

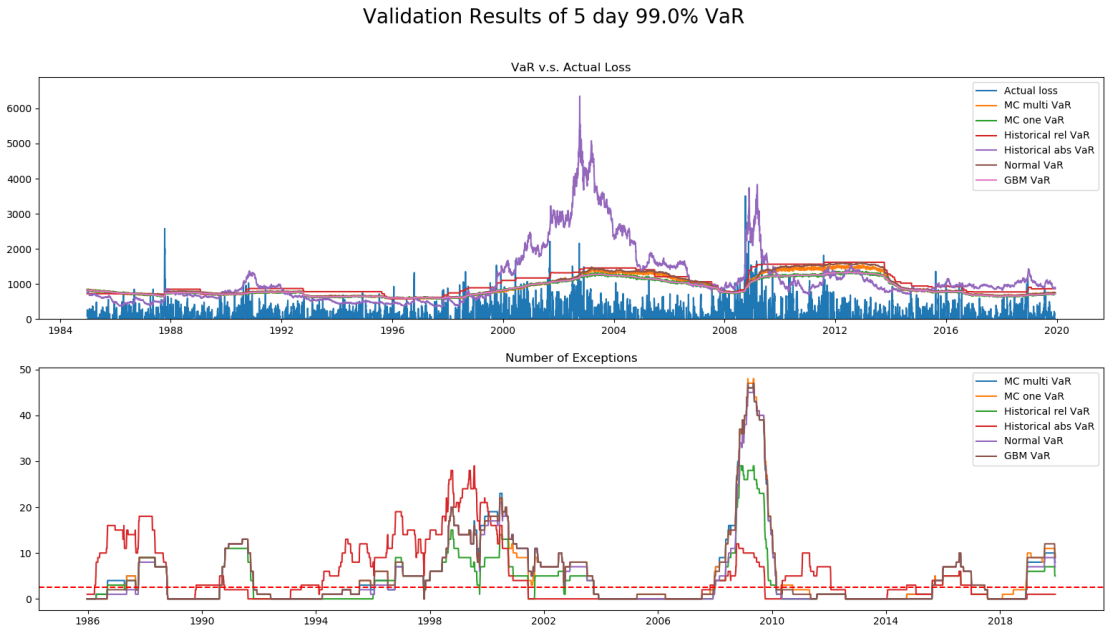

Figure 4.7: Validation Results

Users may adjust the plot size and style or save the plot figures to local drive by clicking the toolbar icons at the lower left corner of the plot window. The plot will be temporarily stored in the system until the next calculation.

4.2.7 Saving

Users can save all the calculation results by clicking the 'Save' button. The system will create a local folder named after current date time and store three CSV files in it. These CSV files are the detailed results of VaR and ES calculation in the form of data frame, as well as the result of validation data frame. The data will be sorted in reversed order to help the users easily access the most up-to-date data. One may read the data using other statistical tools for future reference.

5 Model Validation

Model validation evaluates potential limitations and assumptions in the risk calculation system, and assesses their possible impact. It plays a critical role in financial risk management as it provides essential information on the model risk. Solid model validation can facilitate decision-making process because it proves the reliability of the model. For VaR modeling, the most commonly used validation approach is backtest.

5.1 Backtest Methodology

To evaluate how estimated VaR fits the real scenario, one can perform a backtesting using the number of exceptions within a period. Exceptions stand for the situations where the actual loss in a portfolio violates the calculated VaR at that date. For instance, on date i, the VaR on that date is VaR(i). The actual observed 5-day loss would be Price(i) - Price($i + 5$). By comparing a 252-day actual loss to VaR, the number of exceptions within one year is obtained. The result can either be compared with the expected number of exceptions, or be applied to the Basel VaR backtesting framework.

5.2 Test Plan

In order to test the accuracy of our model and the feasibility of the implementation, a detailed test plan is listed below:

1. Test whether the model correctly estimates GBM parameters
2. Test whether the model generates correct backtest results
3. Test whether the model can work properly for different investment periods
4. Test whether the model works properly for exponentially equivalent parameters
5. Test whether the model can work properly for various portfolio positions, specifically:
    - Stocks only case

- Options only case
- Mixed stocks and options case

6. Test the model can work properly for long and short portfolios, specifically:
   - Long only portfolio
   - Short only portfolio
   - Mixed long and short portfolio
7. Test model sensitivity:
   - VaR / ES level
   - Window Length
   - Horizon period
8. Test numerical stability

5.3 Test Results

In this section, we perform extensive tests and experiments to ensure our model and software works correctly and efficiently. Due to the large number of test scenarios, the input parameters are set to 'default' without any specifications, which is the same as the default values on software calculation page. Default parameters are VaR Level = 0.99, ES Level = 0.975, Portfolio Value = 10000, Window Length = 5, Horizon Period = 5, Estimation Method = 'Window' and maximum investment time period.

5.3.1 Correctness of GBM Calibration

To test the correctness of GBM parameter calibration, we test our calibration function on a synthetic stock, which is simulated using GBM at fixed drift = 0.2 and sigma = 0.1. By adding the synthetic stock to the stock data frame and applying calibration functions, we acquire result as follows:

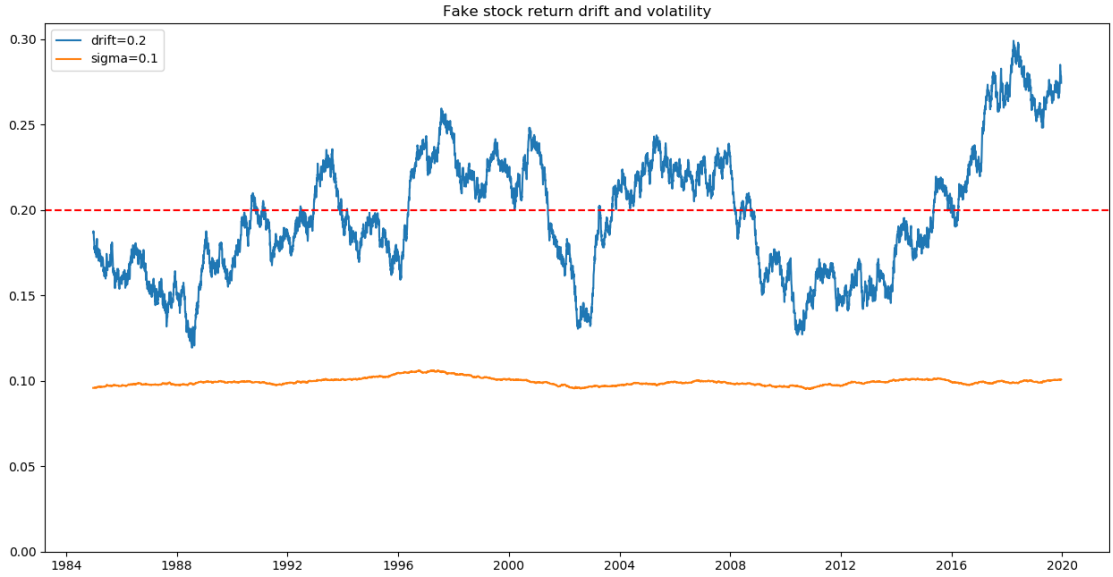

Figure 5.1

The calibration in figure 5.1 shows that the sigma is estimated correctly at around 0.1, while drift, though very volatile, is around 0.2 on average. In fact, as we remove the synthetic sigma by setting it to zero, the calibrated drift becomes a straight line at 0.2 as expected, which means our model in calibrating GBM parameters is totally correct.

5.3.2 Correctness of Validation

To prove the validation result is correct, we reproduce the stock portfolio used in course assignment, which is 175 shares of F and 49 shares of XRX. We use default parameters as other inputs. The result is shown in figure 5.2

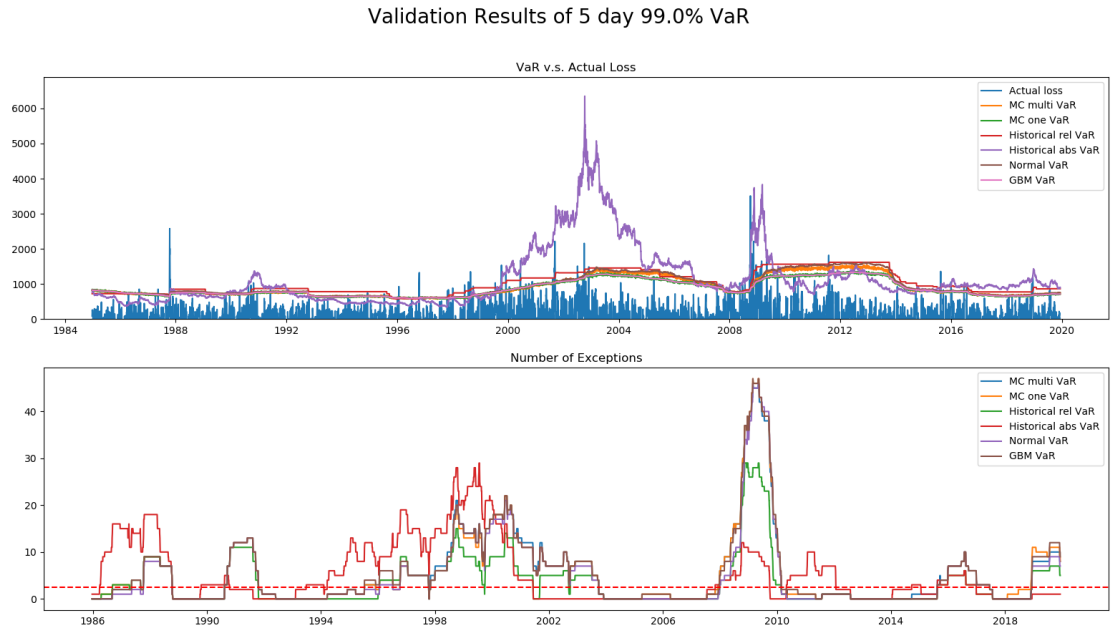

Figure 5.2

The VaR vs. actual loss seems consistent with the number of exceptions trend in the validation. Also the plot successfully reproduces the scenario in assignment, which again concludes that our backtest model is correct.

5.3.3 Changing Time Period

In order to test the effect of changing time period, we use the same portfolio and input parameters as in the correctness of validation section, except that the time period is randomly set to 1999 $\sim$ 2009. The calculated VaR is shown below in figure 5.3.

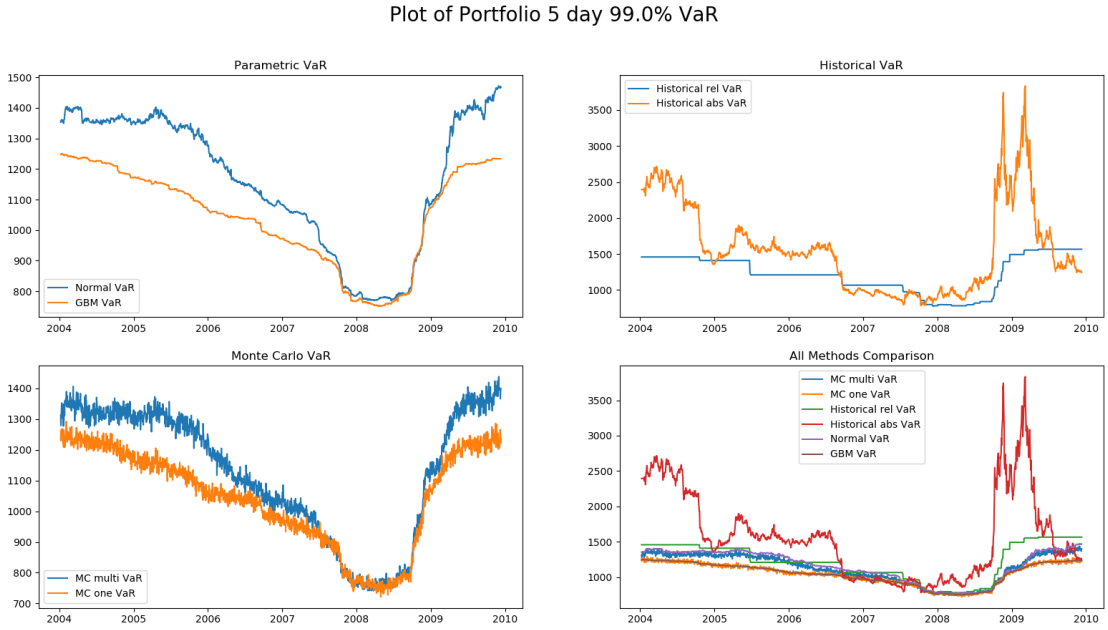

Figure 5.3

The VaR plot in figure 5.3 is exactly a fraction of VaR in figure 5.2, which proves our model correctly adjusted to the changing time periods.

5.3.4 Exponentially Equivalent Parameter

For the same settings, we change the parameter estimation method to exponentially equivalent. The result is shown in figure 5.4.

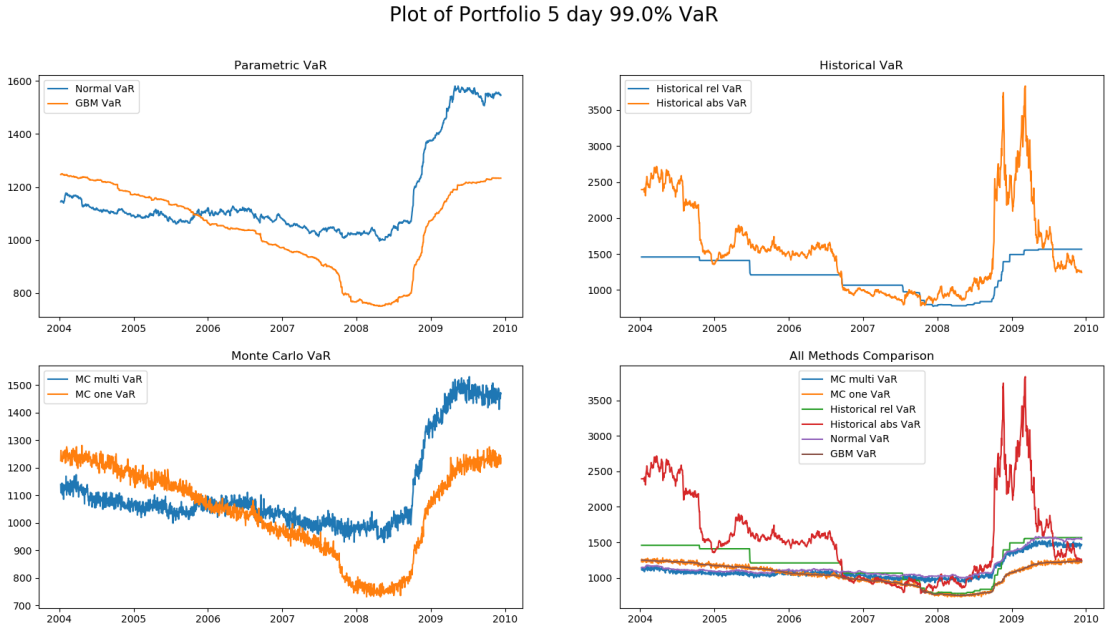

Figure 5.4

Compared with the window sized method in figure 5.3, the parametric and MC VaR changes while historical VaR remains the same. The exponentially equivalent method in this case works properly.

5.3.5 Changing Portfolio Positions

- Stocks only

  We choose MSFT, IBM and KO as the underlying stocks, with default parameters. The system works properly with a valid result. The result is in figure 5.5 and figure 5.6.

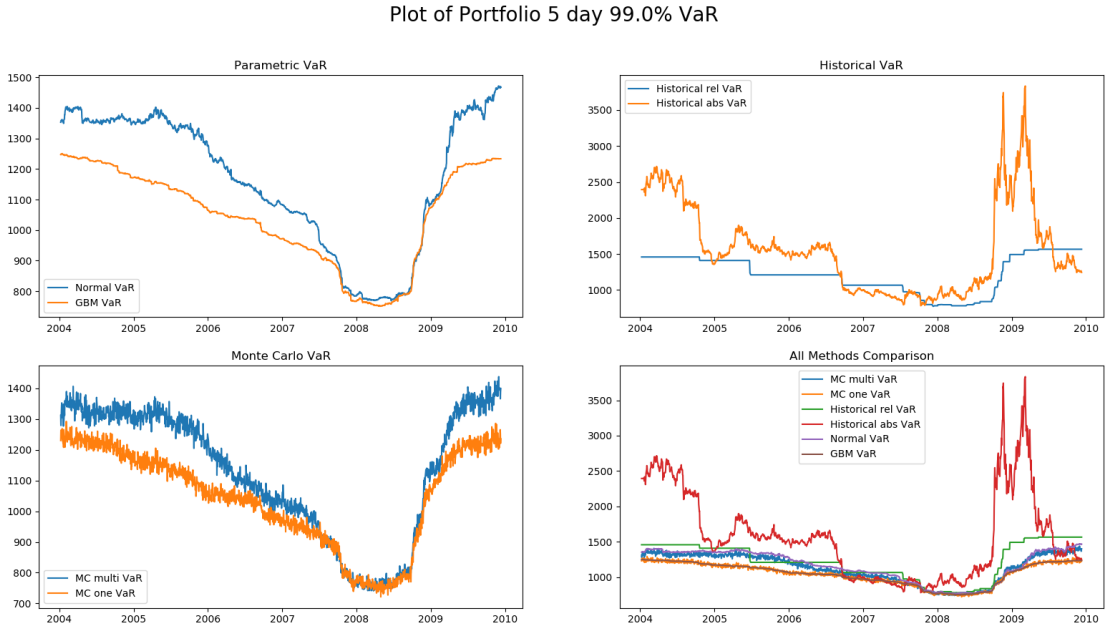

Figure 5.5

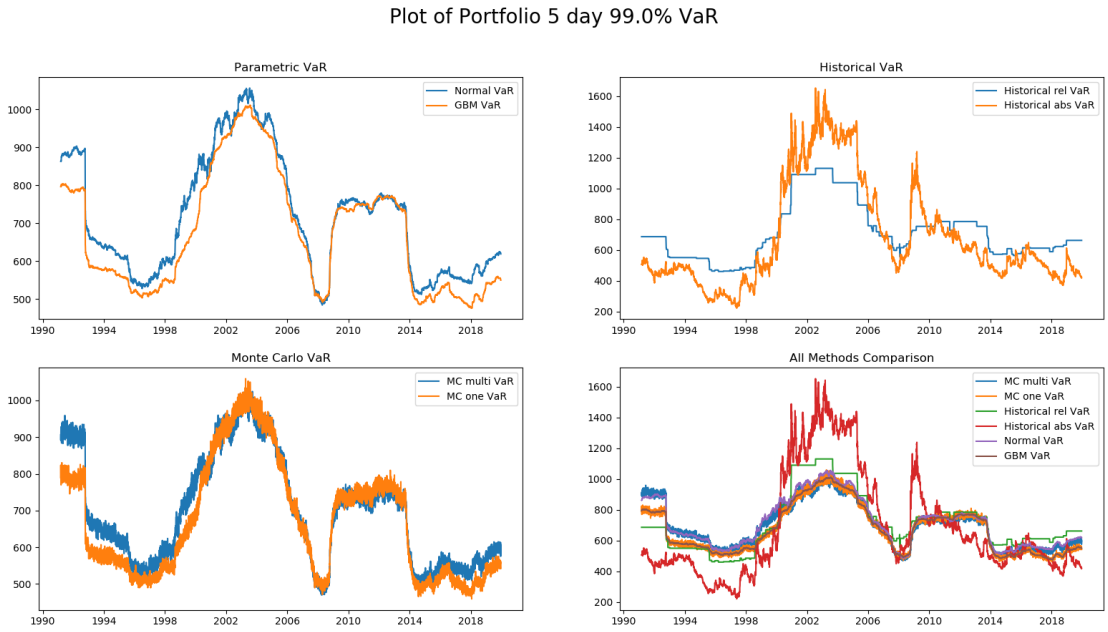

Figure 5.6

- Options only

We choose AAPL call option with 6-month maturity and GE put option with 3-month maturity, with default parameters. The system works properly with a valid result. The result is in figure 5.7 and figure 5.8.

Note that the plot of stock's historical price is empty because there is no stock in the portfolio. With only options, the portfolio has a large sigma and the result for Monte Carlo methods becomes highly volatile.

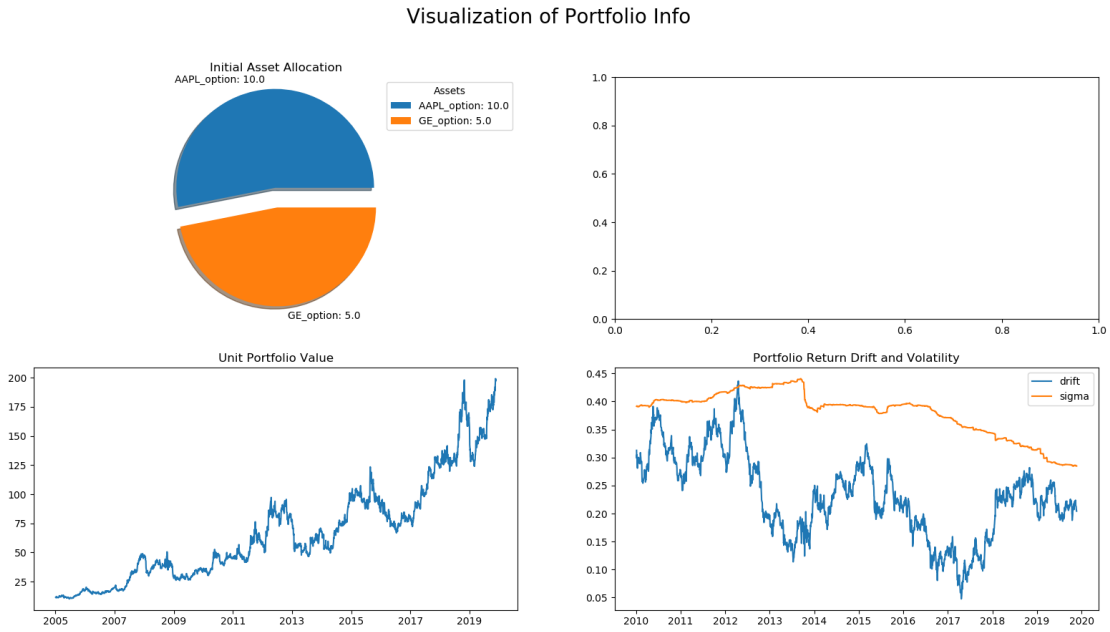

Figure 5.7

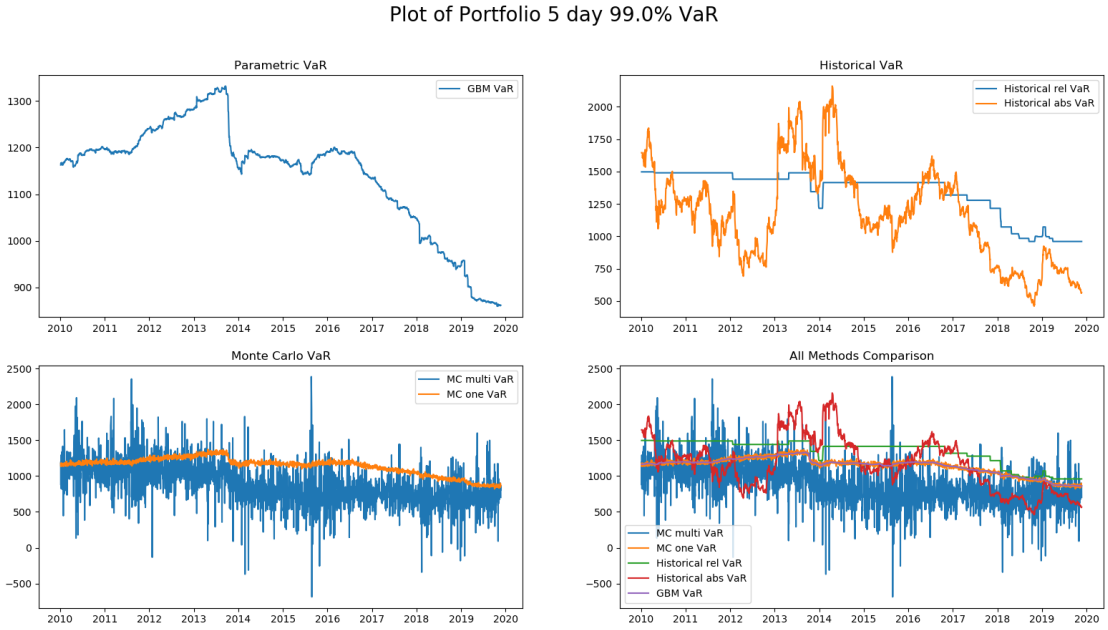

Figure 5.8

- Mixed stocks and options

  We choose XOM, T, BAC as stocks, NVDA 12-month call and CVS 3-month put as options, with default parameters. The system works properly with a valid result. The result is in figure 5.9 and figure 5.10.

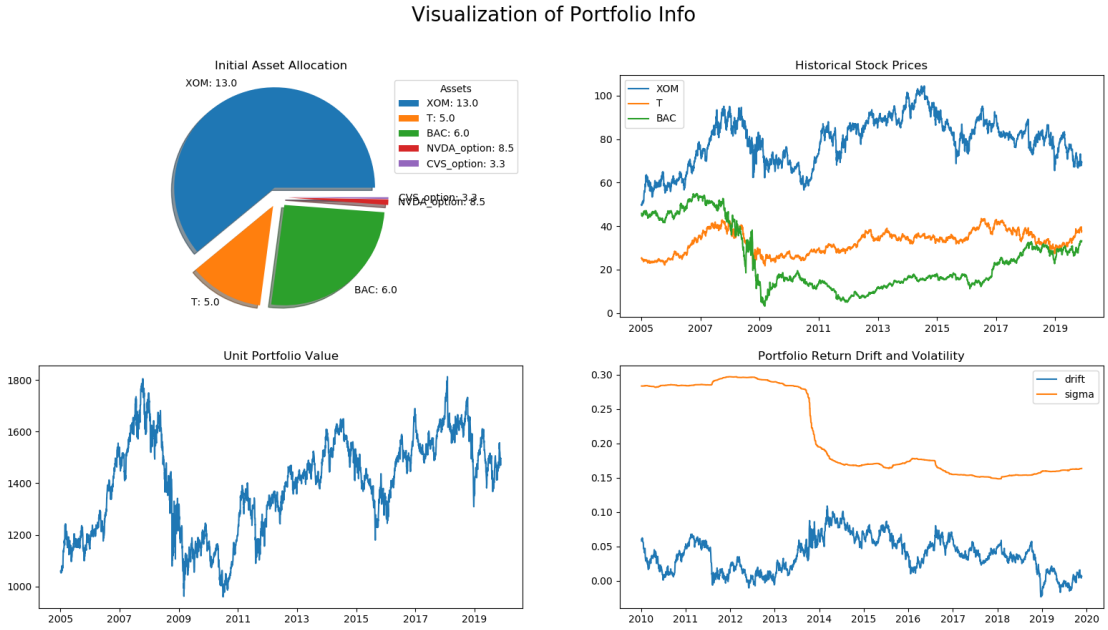

Figure 5.9

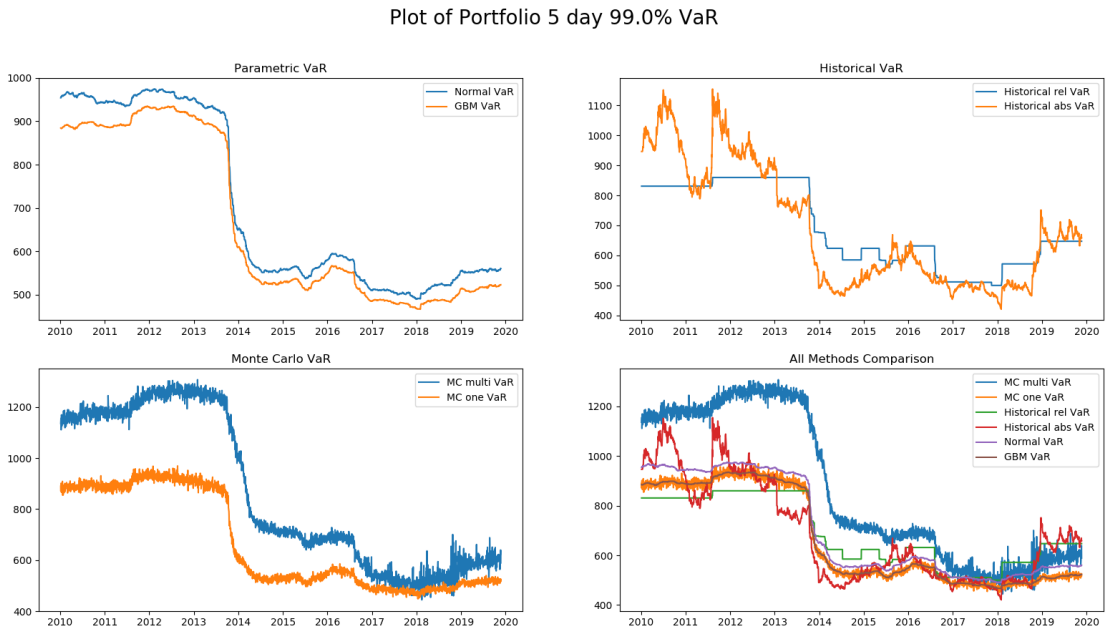

Figure 5.10

5.3.6 Long Short Portfolios

- Long only portfolio

  We randomly pick multiple stocks and options including C, K, BA, TSLA, ATVI, AAL, AIG, EBAY, JPM, GE, NVDA, CVS in the portfolio, each with a positive number of shares. We use ES plot as an example without loss of generality. The result is in figure 5.11 and figure 5.12.

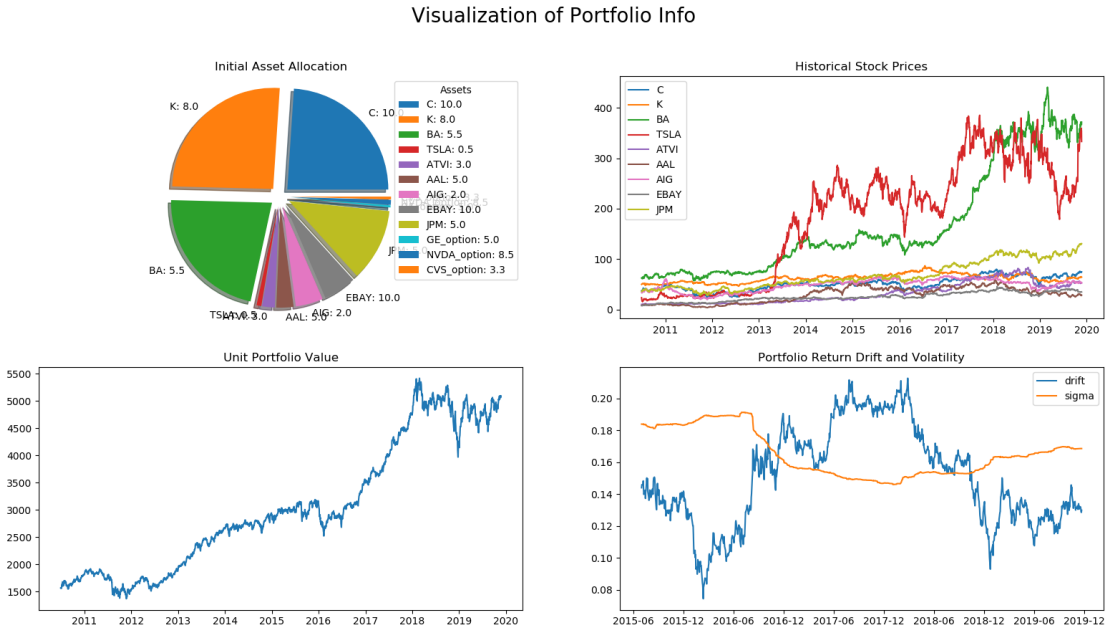

Figure 5.11

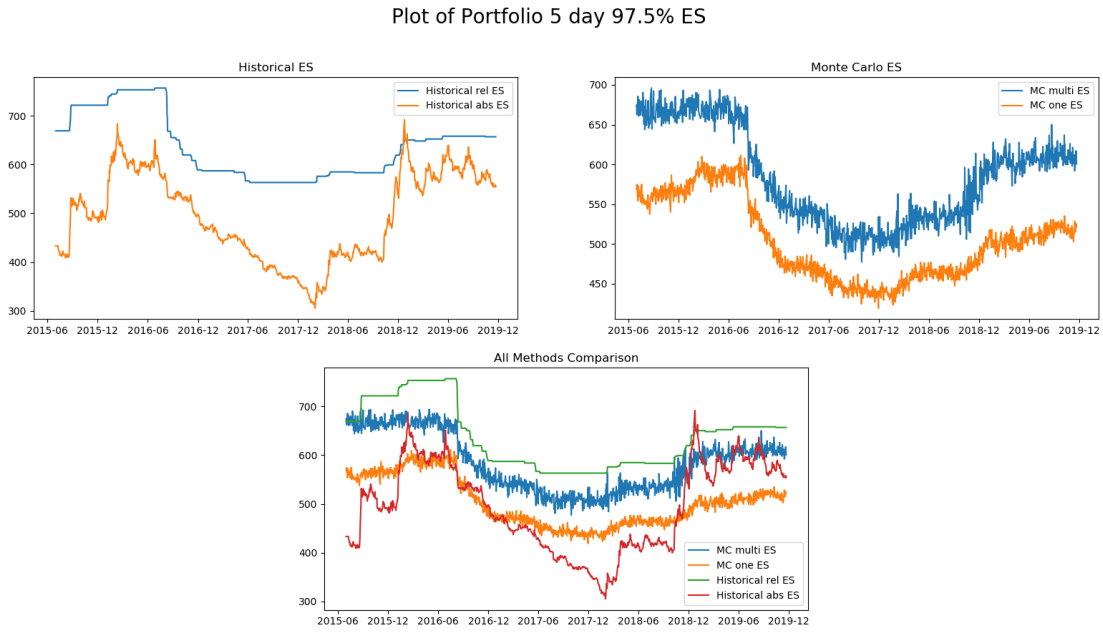

Figure 5.12

- Short only portfolio

  Next we flip the positions of assets in the previous section. Again we use ES plot as an illustration to show our system works. The result is in figure 5.13 and figure 5.14. Note that a negative portfolio value stands for a short portfolio.

  The comparison demonstrated in figure 5.12 suggests that the shape of ES in two portfolios are similar, but short portfolio has a higher absolute value. This is consistent with the fact that for a short portfolio, the loss is unbounded. As a result, it results in a higher expected value as well as calculated ES.

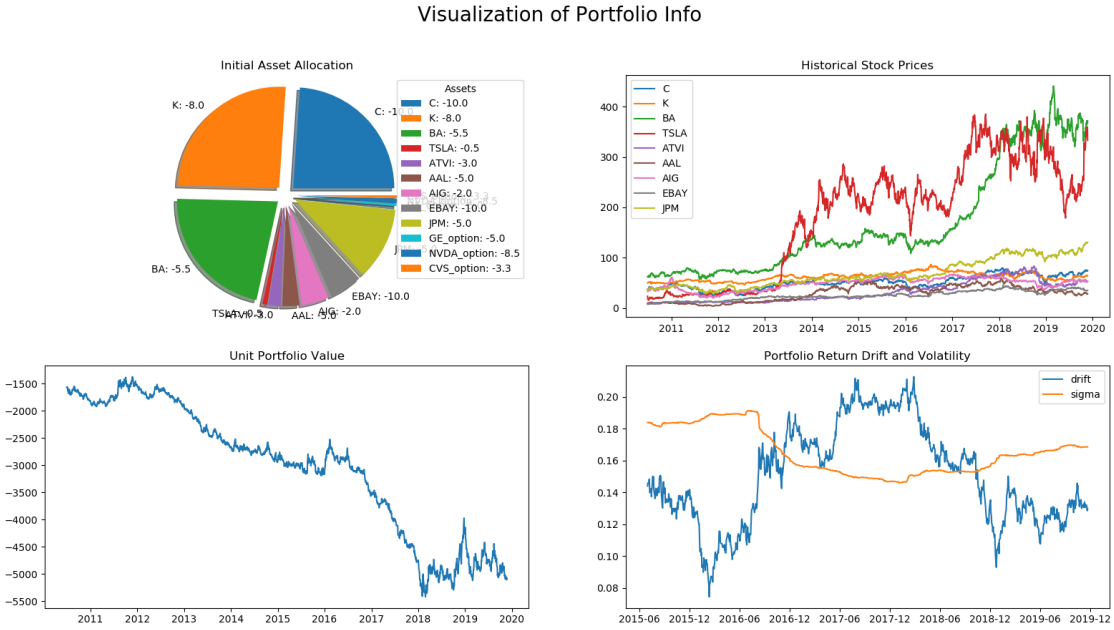

Figure 5.13

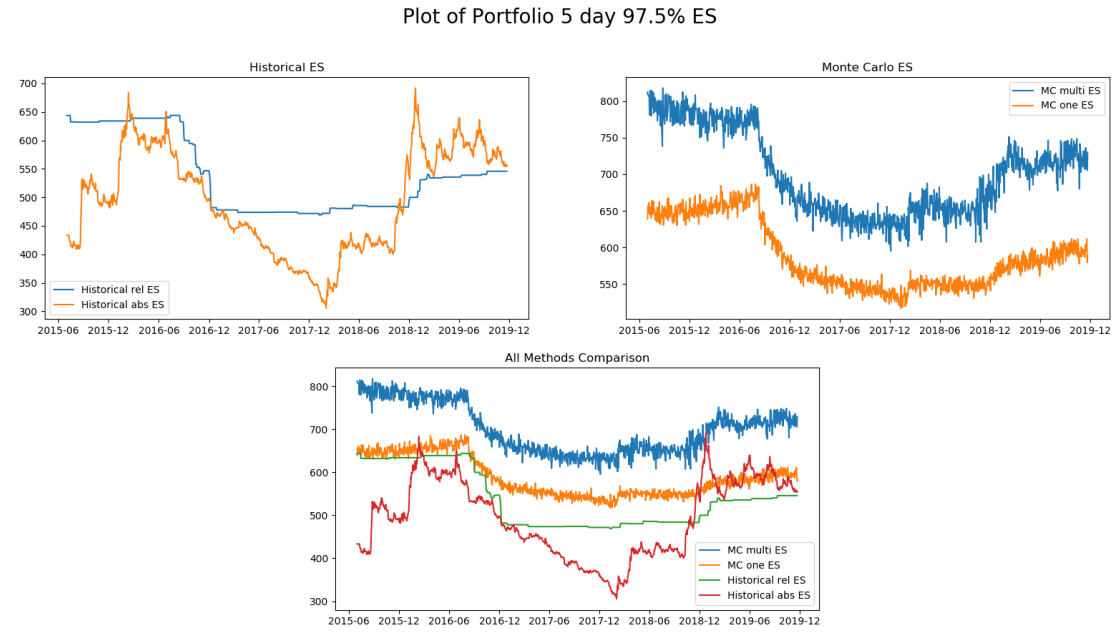

Figure 5.14

- Mixed long and short portfolio

We create another portfolio with random positive and negative positions in stocks and options including AMD, TSLA, VZ, GOOG, WFC, KEY, FDX, NKE, AMZN. The results are shown in figure 5.15 and figure 5.16.

Note that if a portfolio contains both long and short position in different stocks, the parametric normal VaR and MC Multi VaR / ES will generally become larger and more volatile in certain years. This is due to the hedging effect eliminates some of the volatility for the portfolio as a whole in overall bull or bear markets, as suggested by MC ES plot in figure 5.16.

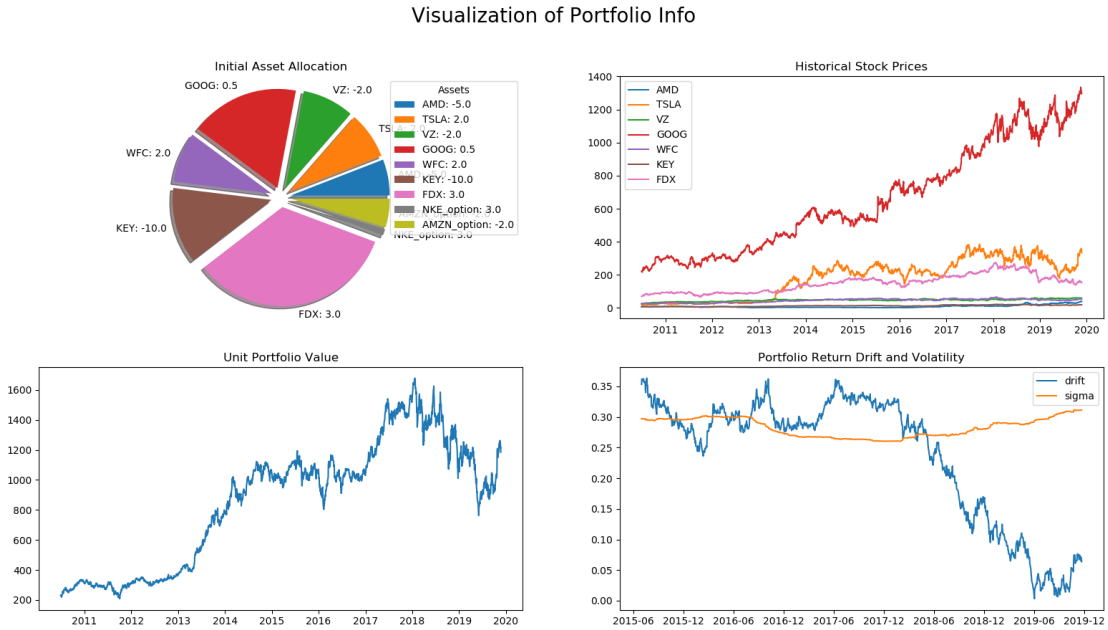

Figure 5.15

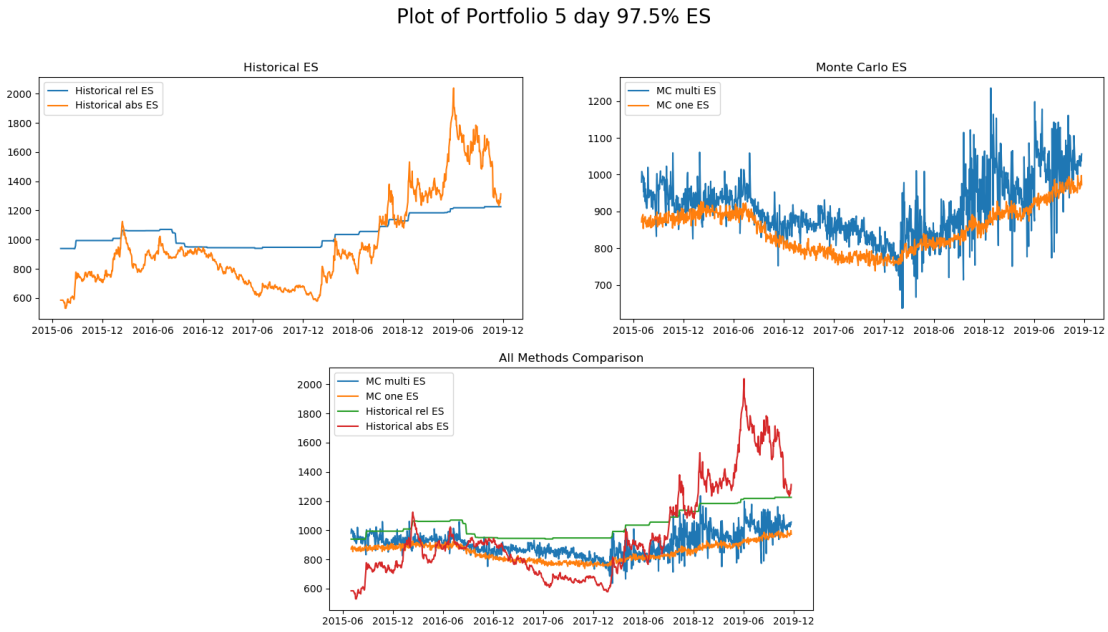

Figure 5.16

5.3.7 Model Sensitivity

To test model sensitivity, we choose a portfolio consisting of the following stocks: MSFT, SNE, AMZN, GOOG, XOM, T, BAC, BA and options taking AAPL, GE, NVDA, KO as underlying stocks. We first set all the parameters to default values. Every time we change only one parameter while keeping all the other variables constant.

1. VaR / ES Level

The plot of VaR, ES and backtesting is shown in figure 5.17 to figure 5.28 for 99%, 95%, 90%, 50% VaR accordingly.

a. 99% VaR

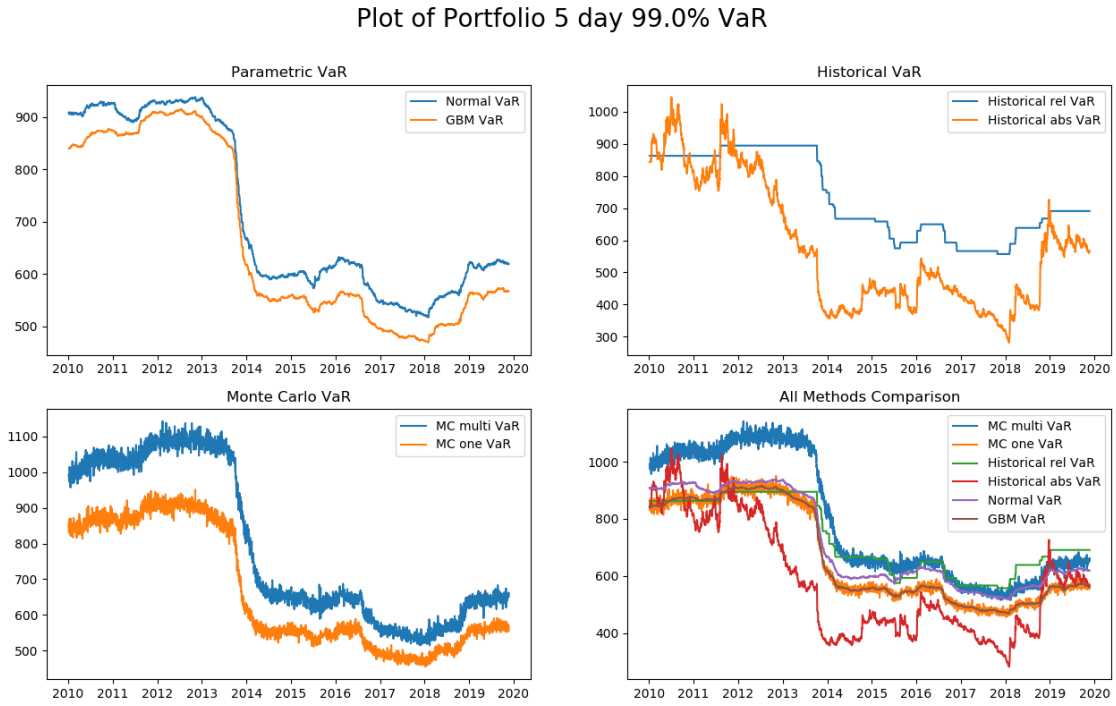
Plot of Portfolio 5 day 99.0% VaR
Parametric VaR
Normal VaR
GBM VaR
Historical VaR
Historical rel VaR
Historical abs VaR
Monte Carlo VaR
MC multi VaR
MC one VaR
All Methods Comparison
MC multi VaR
MC one VaR
Historical rel VaR
Historical abs VaR
Normal VaR
GBM VaR

Figure 5.17

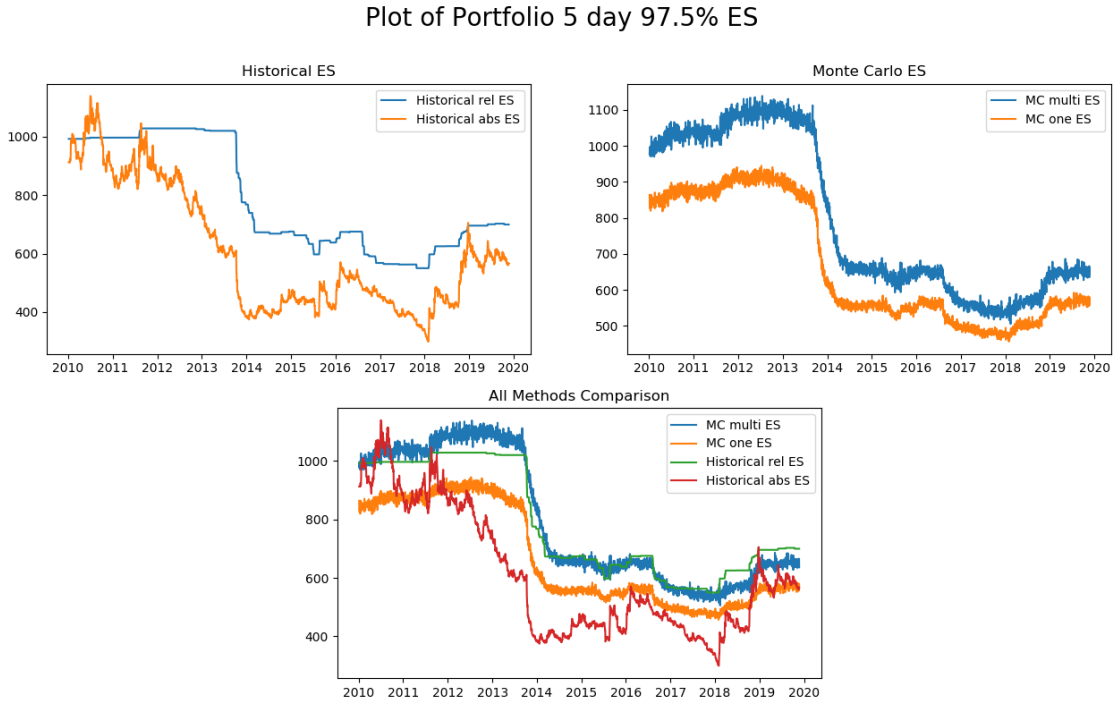
Plot of Portfolio 5 day 97.5% ES
Historical ES
Historical rel ES
Historical abs ES
Monte Carlo ES
MC multi ES
MC one ES
All Methods Comparison
MC multi ES
MC one ES
Historical rel ES
Historical abs ES

Figure 5.18

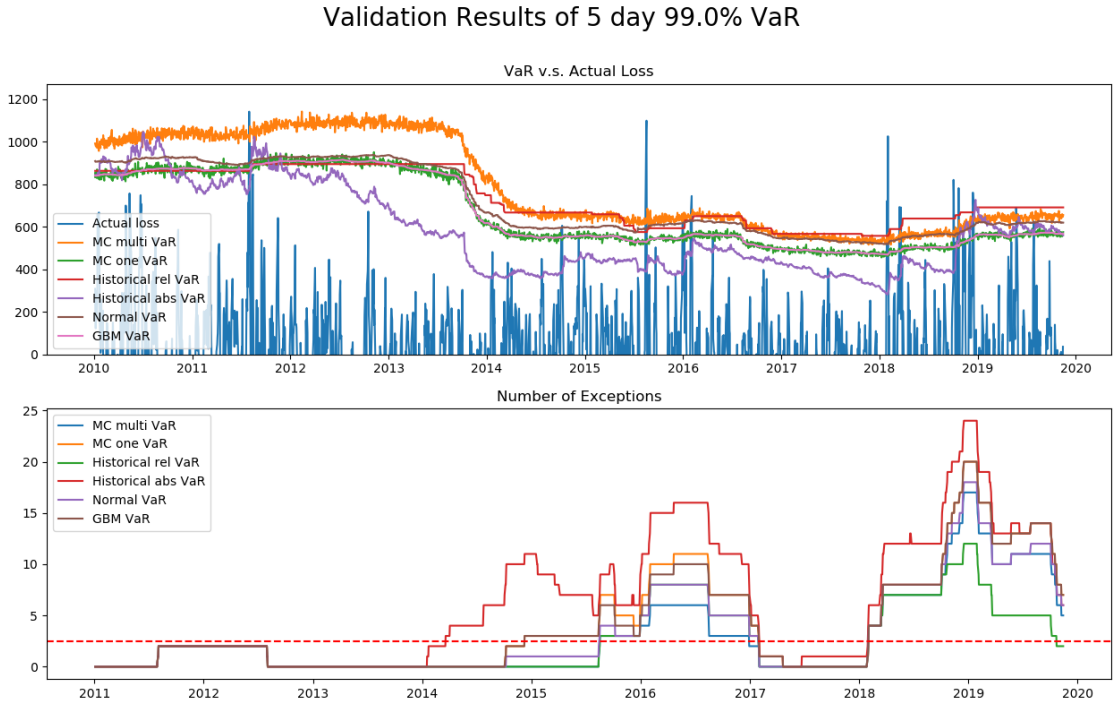

Figure 5.19

b. 95% VaR

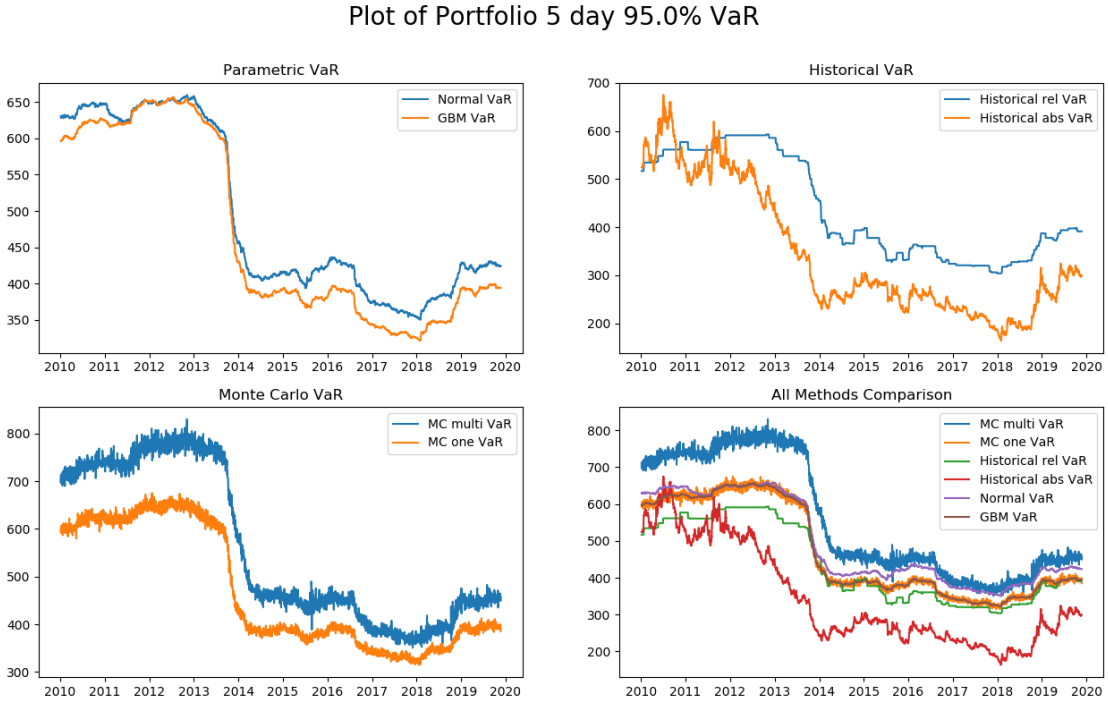
Plot of Portfolio 5 day 95.0% VaR
Parametric VaR
Normal VaR
GBM VaR
Historical VaR
Historical rel VaR
Historical abs VaR
Monte Carlo VaR
MC multi VaR
MC one VaR
All Methods Comparison
MC multi VaR
MC one VaR
Historical rel VaR
Historical abs VaR
Normal VaR
GBM VaR

Figure 5.20

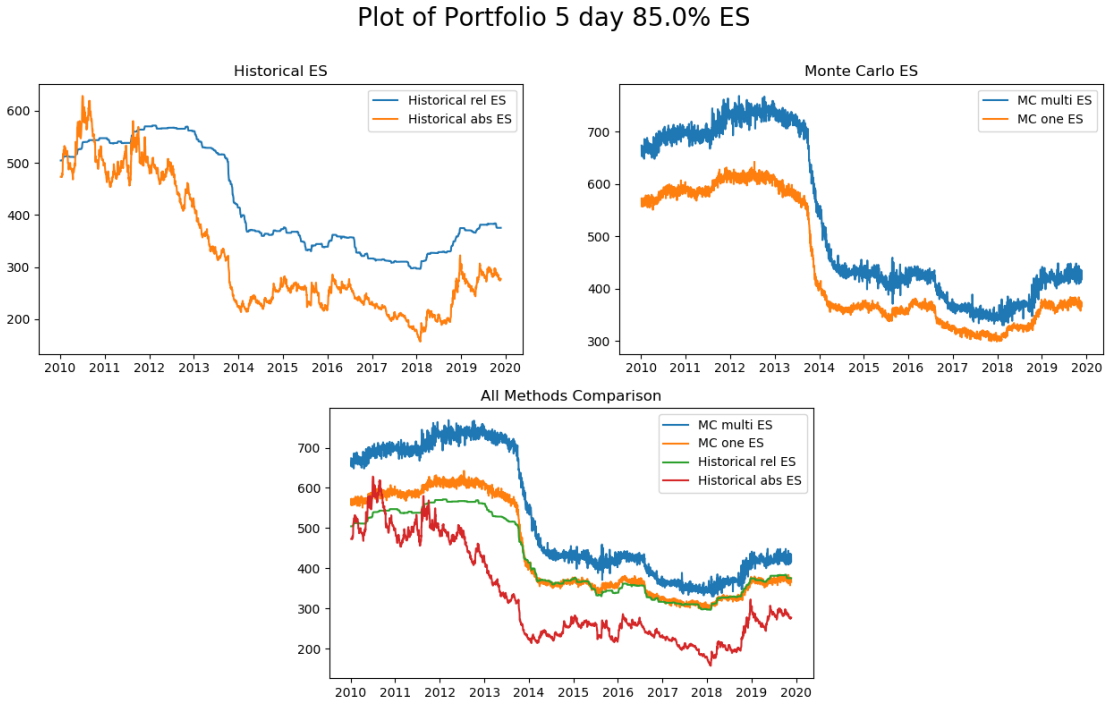
Plot of Portfolio 5 day 85.0% ES
Historical ES
Historical rel ES
Historical abs ES
Monte Carlo ES
MC multi ES
MC one ES
All Methods Comparison
MC multi ES
MC one ES
Historical rel ES
Historical abs ES

Figure 5.21

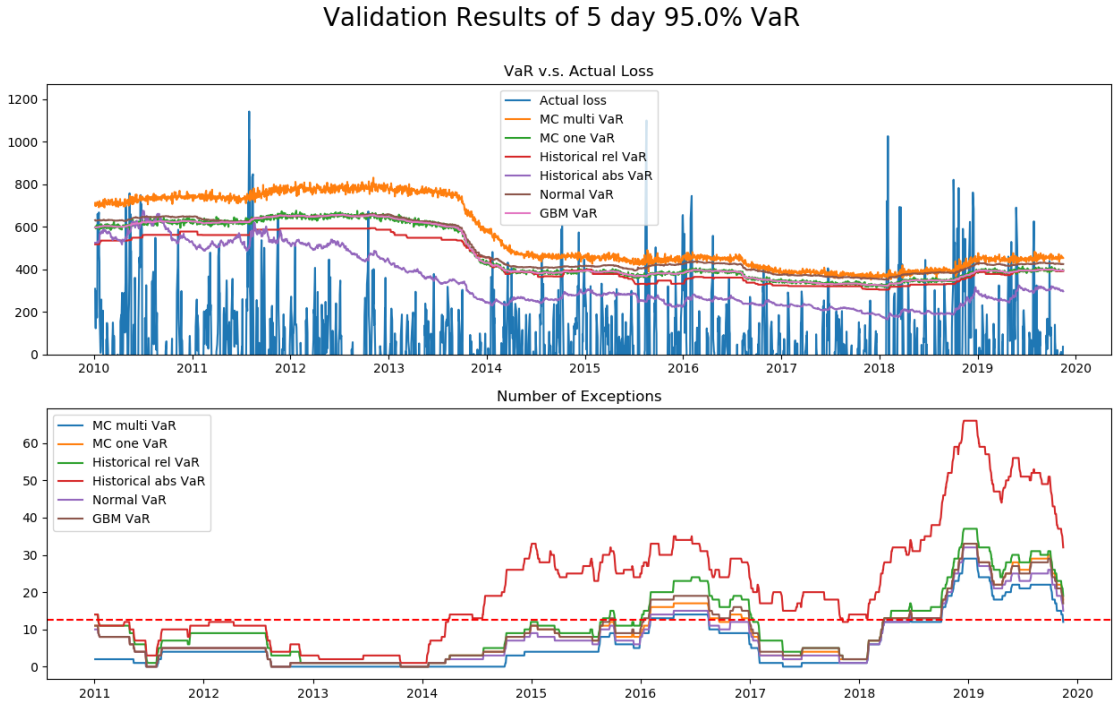

Figure 5.22

c. 90% VaR

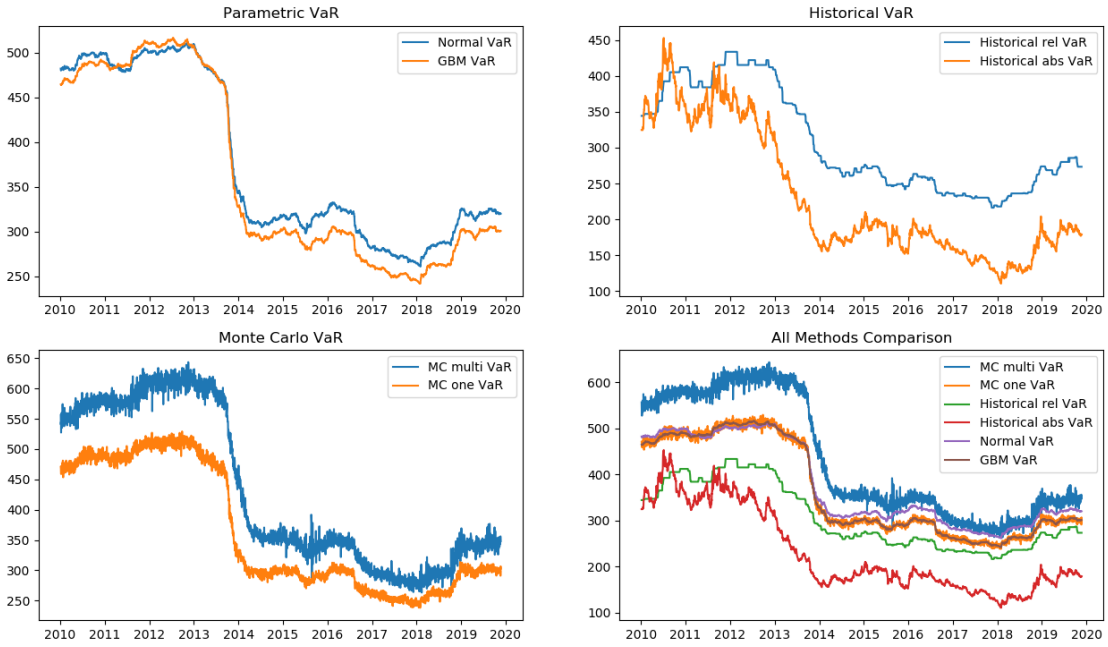
Plot of Portfolio 5 day 90.0% VaR
Parametric VaR
Normal VaR
GBM VaR
Historical VaR
Historical rel VaR
Historical abs VaR
Monte Carlo VaR
MC multi VaR
MC one VaR
All Methods Comparison
MC multi VaR
MC one VaR
Historical rel VaR
Historical abs VaR
Normal VaR
GBM VaR

Figure 5.23

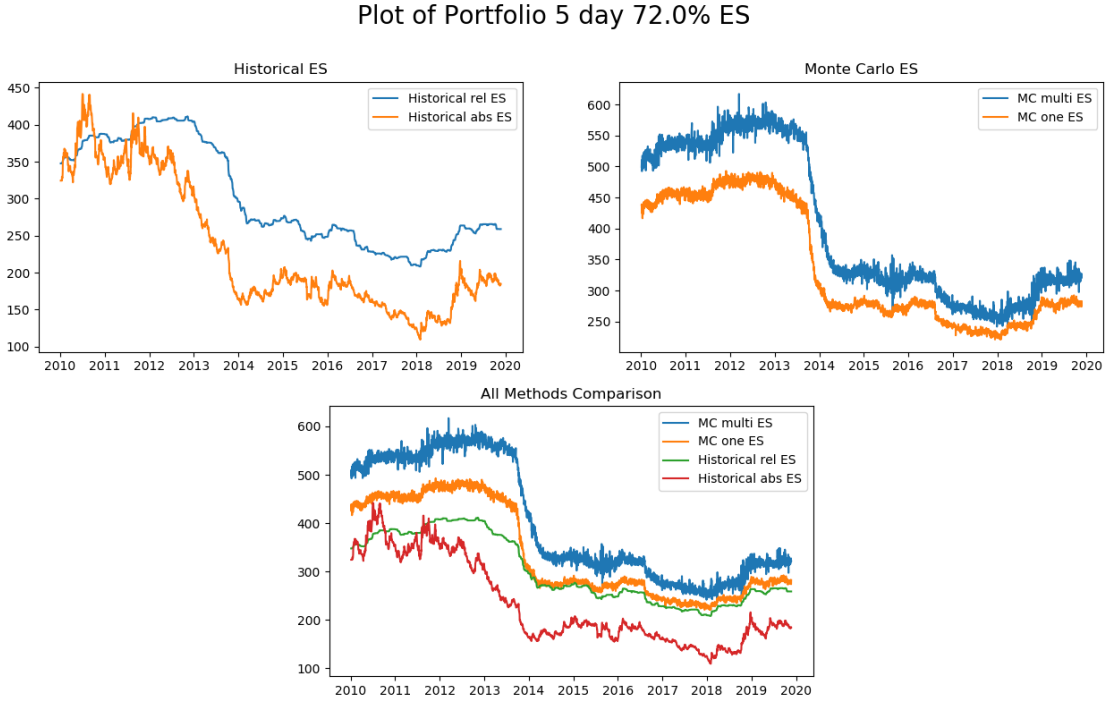
Plot of Portfolio 5 day 72.0% ES
Historical ES
Historical rel ES
Historical abs ES
Monte Carlo ES
MC multi ES
MC one ES
All Methods Comparison
MC multi ES
MC one ES
Historical rel ES
Historical abs ES

Figure 5.24

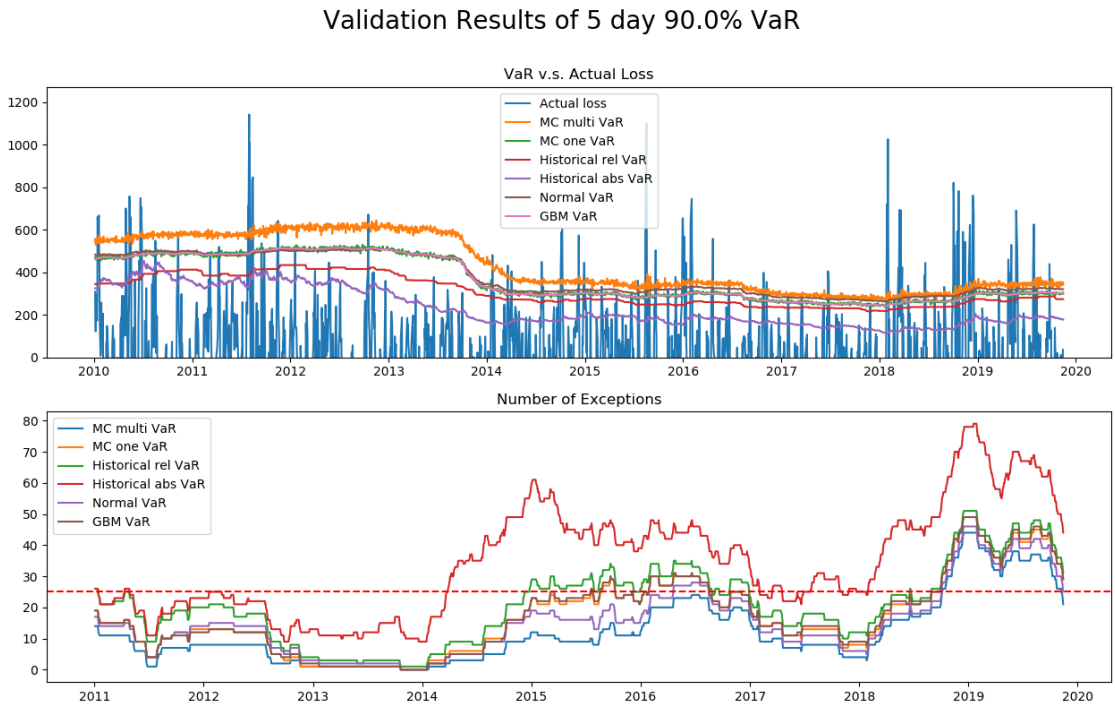

Figure 5.25

d. 50% VaR

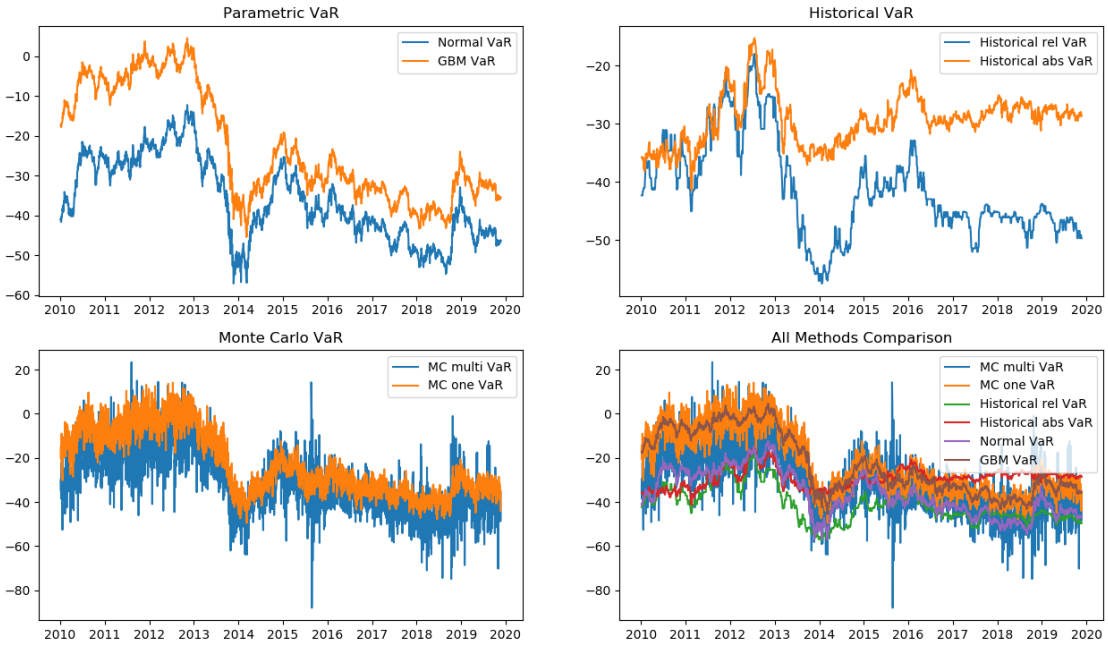
Plot of Portfolio 5 day 50.0% VaR
Parametric VaR
Normal VaR
GBM VaR
Historical VaR
Historical rel VaR
Historical abs VaR
Monte Carlo VaR
MC multi VaR
MC one VaR
All Methods Comparison
MC multi VaR
MC one VaR
Historical rel VaR
Historical abs VaR
Normal VaR
GBM VaR

Figure 5.26

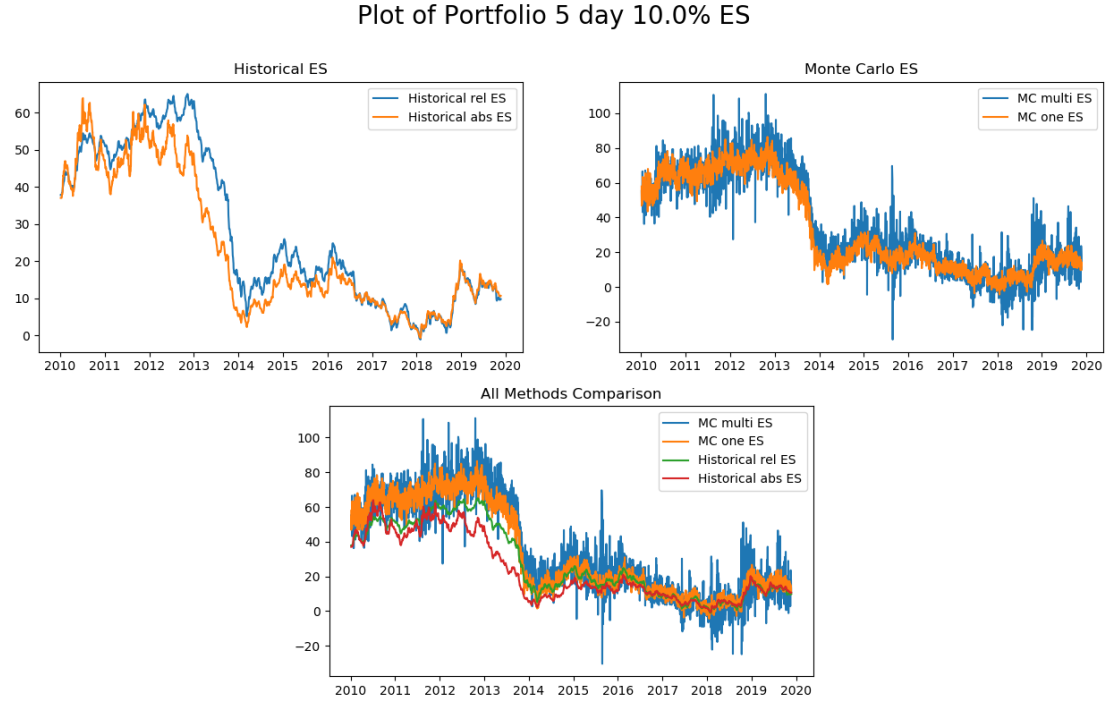
Plot of Portfolio 5 day 10.0% ES
Historical ES
Historical rel ES
Historical abs ES
Monte Carlo ES
MC multi ES
MC one ES
All Methods Comparison
MC multi ES
MC one ES
Historical rel ES
Historical abs ES

Figure 5.27

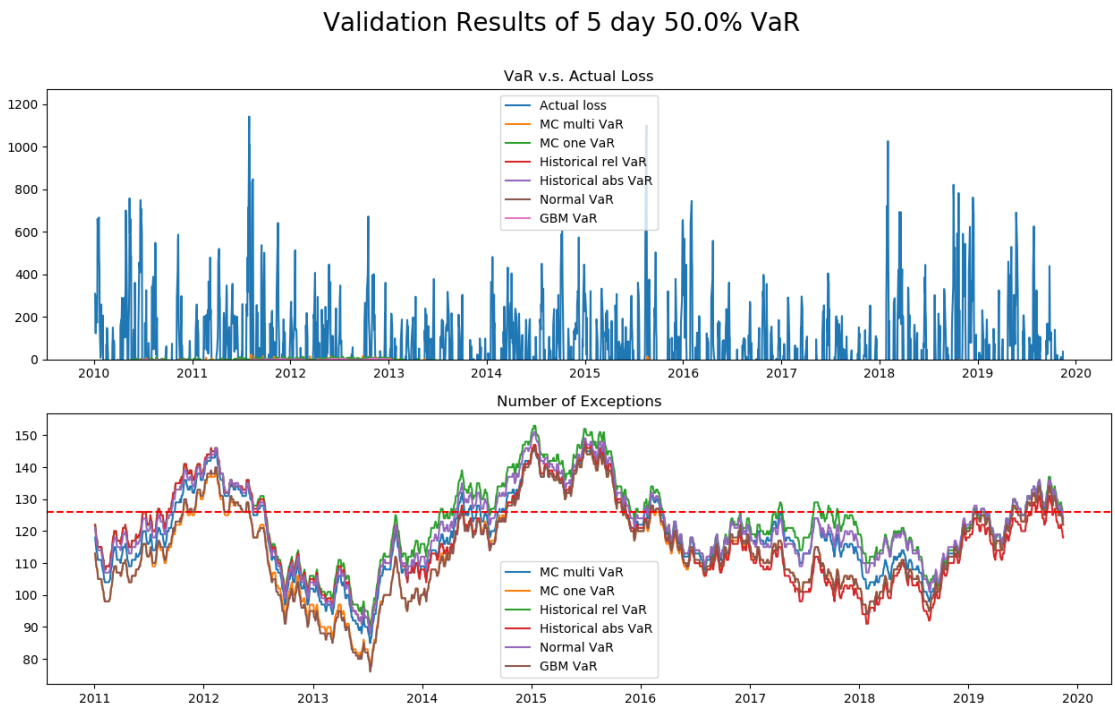

Figure 5.28

From the results above, it can be seen that the portfolio's VaR and ES have similar moving trend for all confidence level. As the level of confidence decreases, the movement becomes less volatile as expected. This proves the validity of our risk calculation system.

2. Window Length

The plot of VaR, ES and backtesting is shown in figure 5.29 to figure 5.37 for 1 year, 3 years and 10 years window length accordingly.

a. 1 year

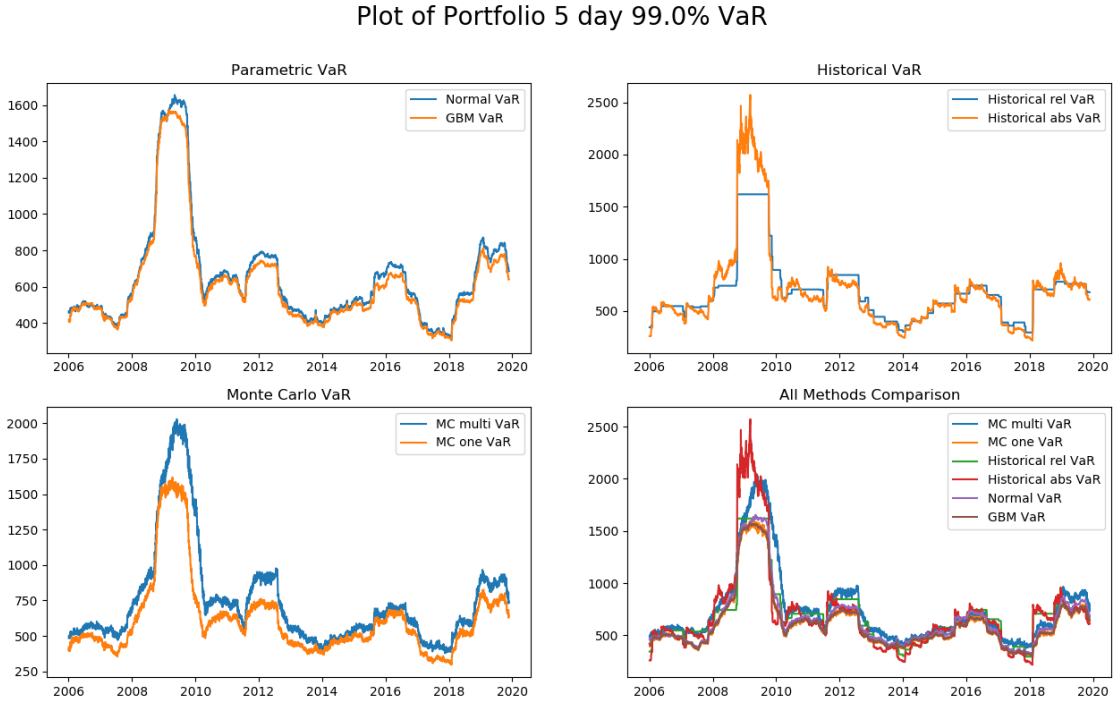
Plot of Portfolio 5 day 99.0% VaR
Parametric VaR
Normal VaR
GBM VaR
Historical VaR
Historical rel VaR
Historical abs VaR
Monte Carlo VaR
MC multi VaR
MC one VaR
All Methods Comparison
MC multi VaR
MC one VaR
Historical rel VaR
Historical abs VaR
Normal VaR
GBM VaR

Figure 5.29

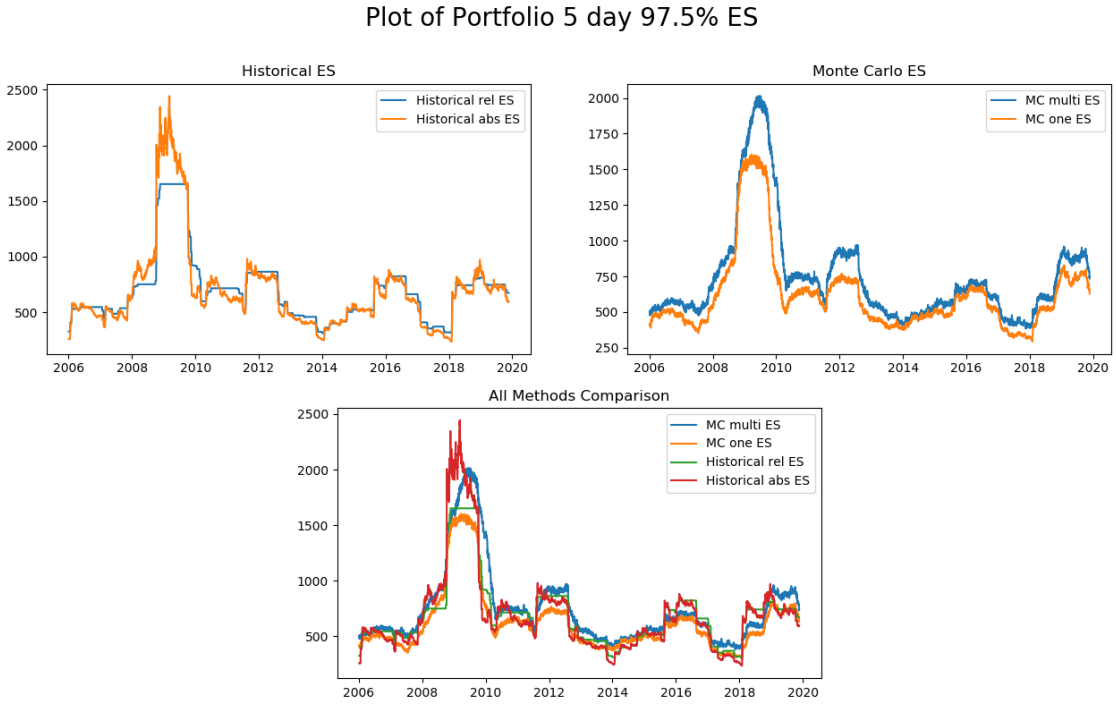
Plot of Portfolio 5 day 97.5% ES
Historical ES
Historical rel ES
Historical abs ES
Monte Carlo ES
MC multi ES
MC one ES
All Methods Comparison
MC multi ES
MC one ES
Historical rel ES
Historical abs ES

Figure 5.30

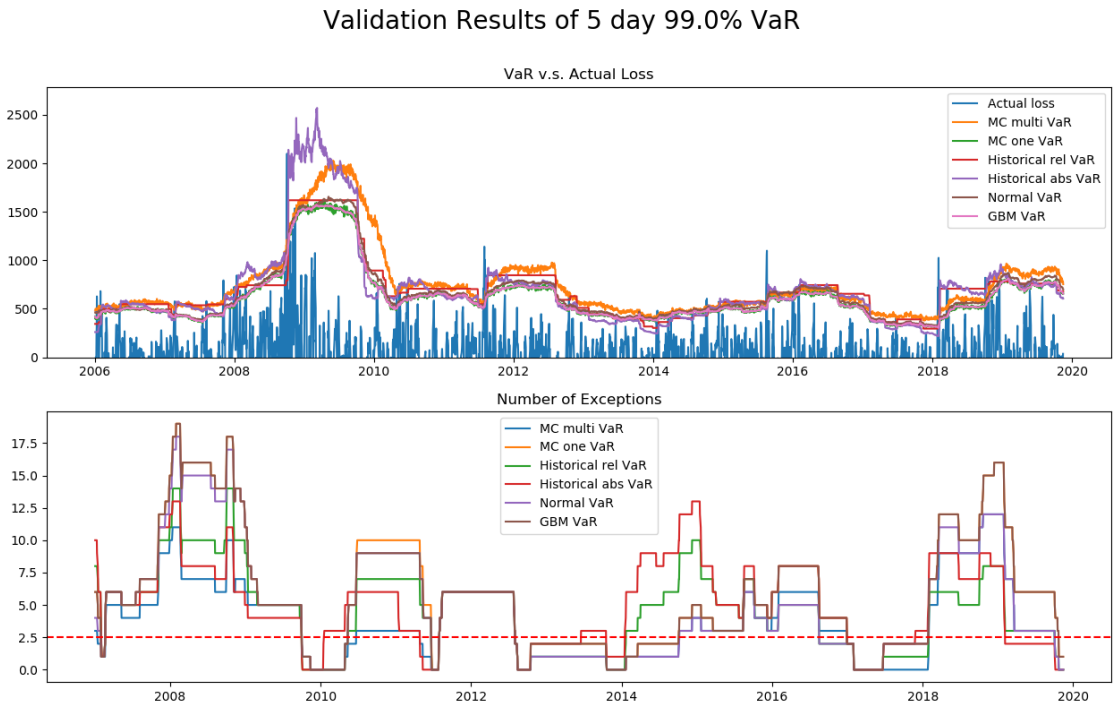

Figure 5.31

b. 3 years

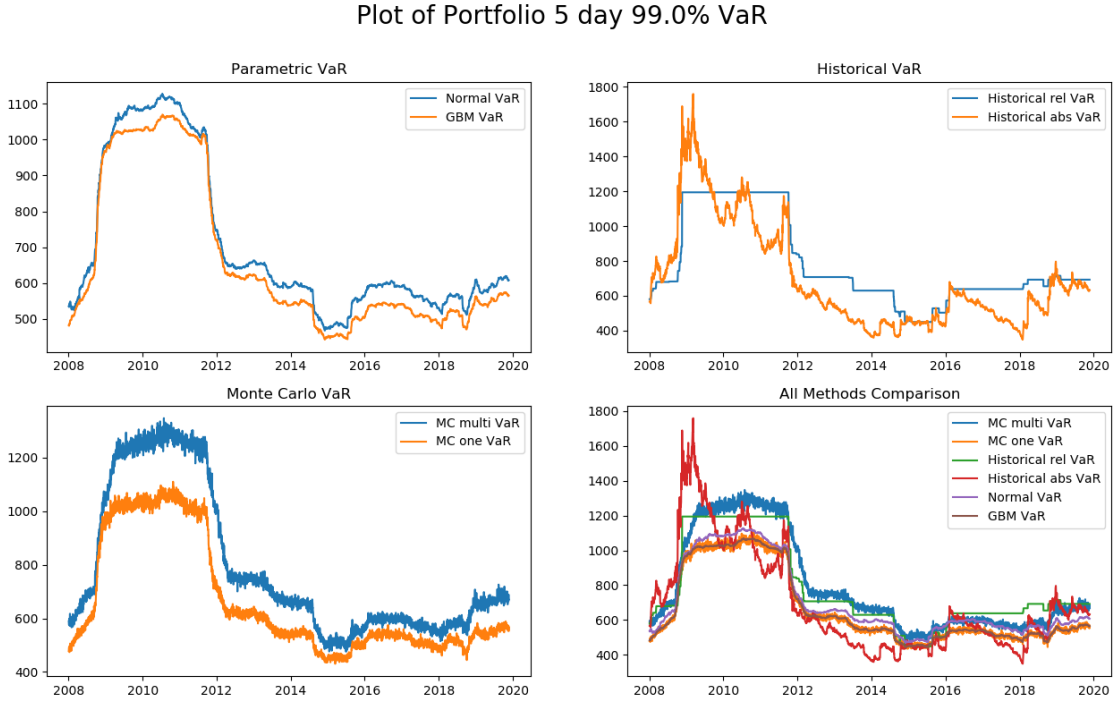
Plot of Portfolio 5 day 99.0% VaR
Parametric VaR
Normal VaR
GBM VaR
Historical VaR
Historical rel VaR
Historical abs VaR
Monte Carlo VaR
MC multi VaR
MC one VaR
All Methods Comparison
MC multi VaR
MC one VaR
Historical rel VaR
Historical abs VaR
Normal VaR
GBM VaR

Figure 5.32

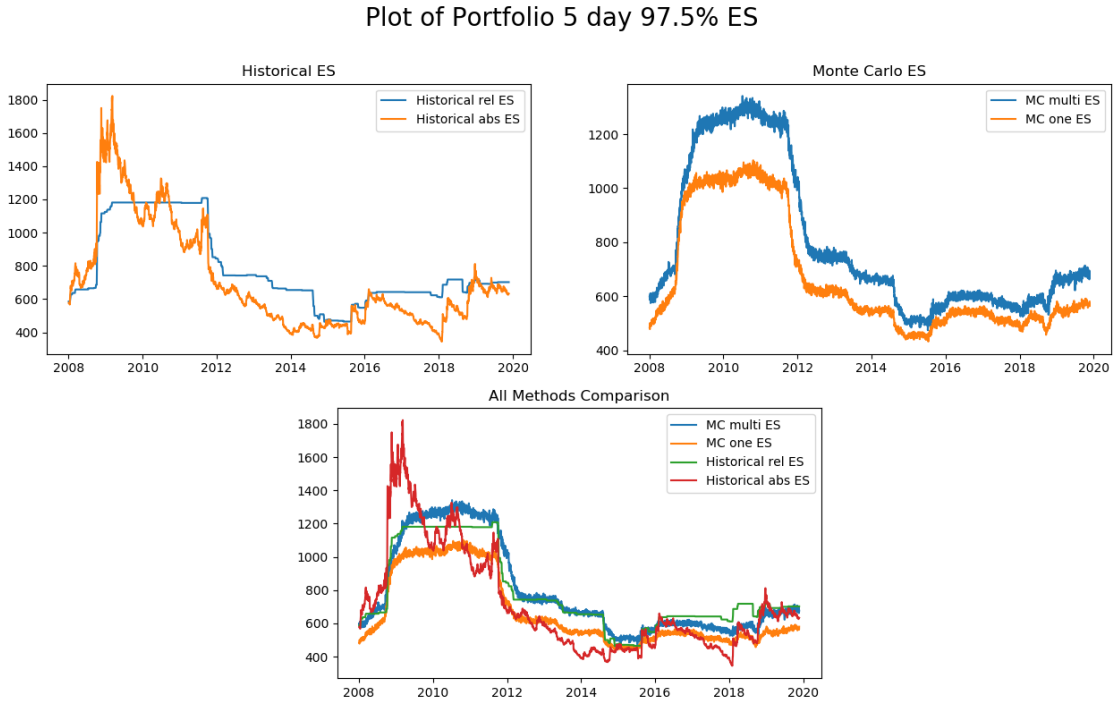
Plot of Portfolio 5 day 97.5% ES
Historical ES
Historical rel ES
Historical abs ES
Monte Carlo ES
MC multi ES
MC one ES
All Methods Comparison
MC multi ES
MC one ES
Historical rel ES
Historical abs ES

Figure 5.33

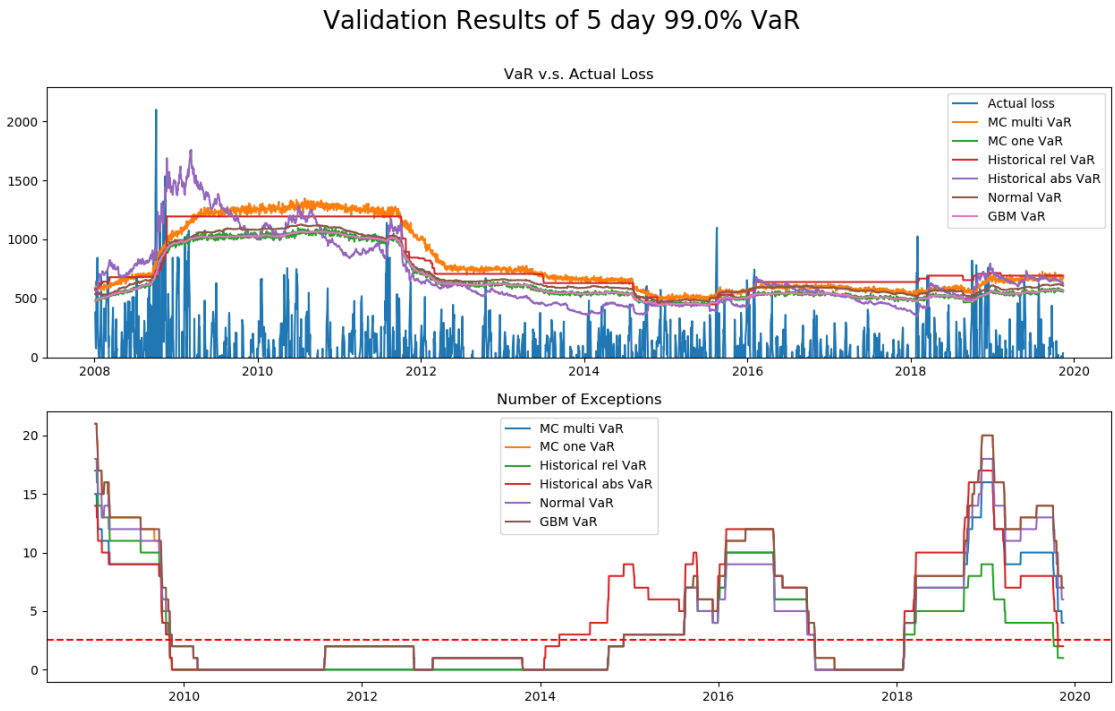

Figure 5.34

c. 10 years

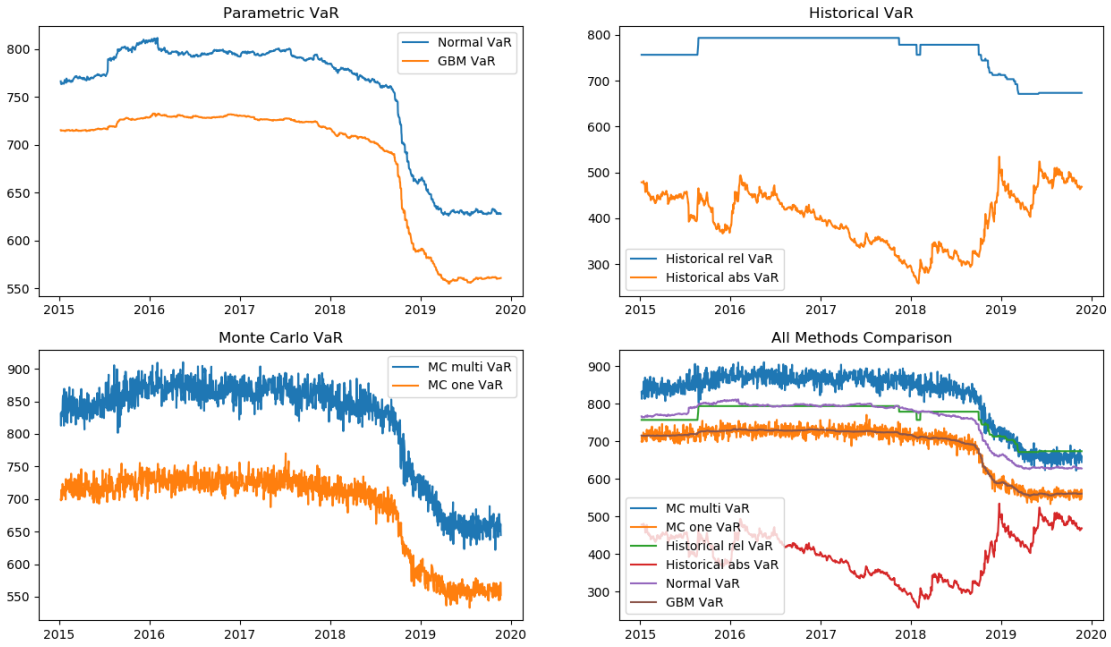
Plot of Portfolio 5 day 99.0% VaR
Parametric VaR
Normal VaR
GBM VaR
Historical VaR
Historical rel VaR
Historical abs VaR
Monte Carlo VaR
MC multi VaR
MC one VaR
All Methods Comparison
MC multi VaR
MC one VaR
Historical rel VaR
Historical abs VaR
Normal VaR
GBM VaR

Figure 5.35

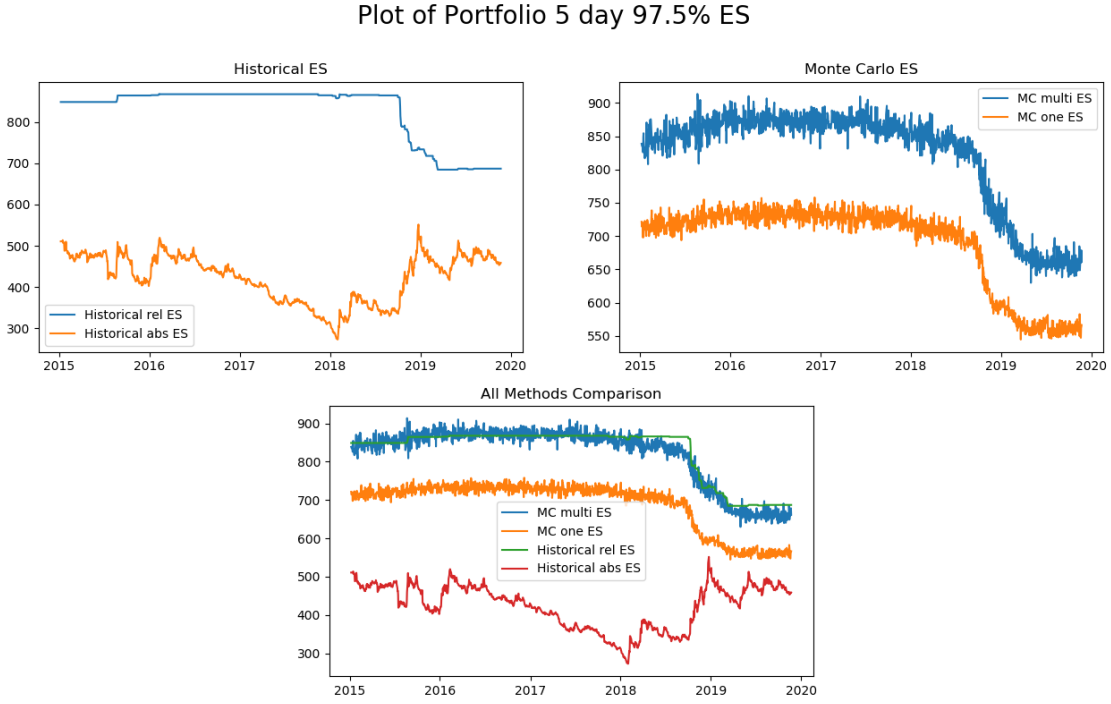
Plot of Portfolio 5 day 97.5% ES
Historical ES
Historical rel ES
Historical abs ES
Monte Carlo ES
MC multi ES
MC one ES
All Methods Comparison
MC multi ES
MC one ES
Historical rel ES
Historical abs ES

Figure 5.36

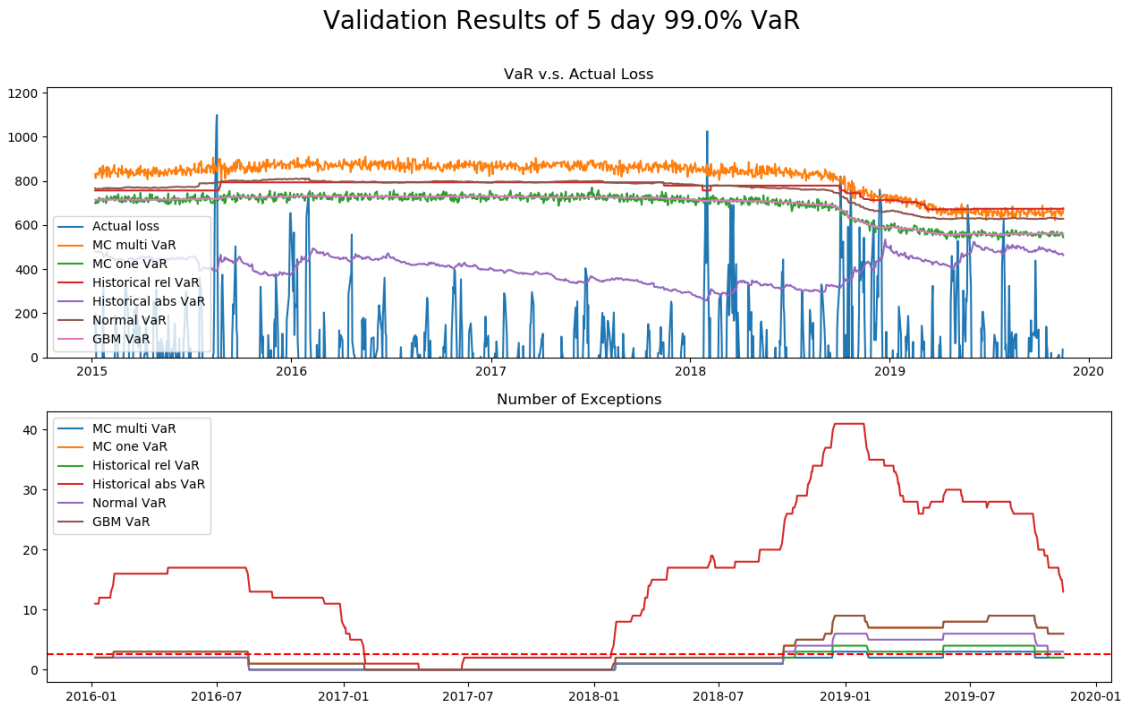

Figure 5.37

As the window length increases, VaR and ES movement become more stationary. In the backtest result, historical VaR using the absolute change method demonstrates high number of exceptions when setting window length to 10 years, validating that this calculation method might not be effective in this case.

3. Horizon

The plot of VaR, ES and backtesting is shown in figure 5.38 to figure 5.46 for 1 day, 1 week and 1 month horizon period accordingly.

a. 1 day

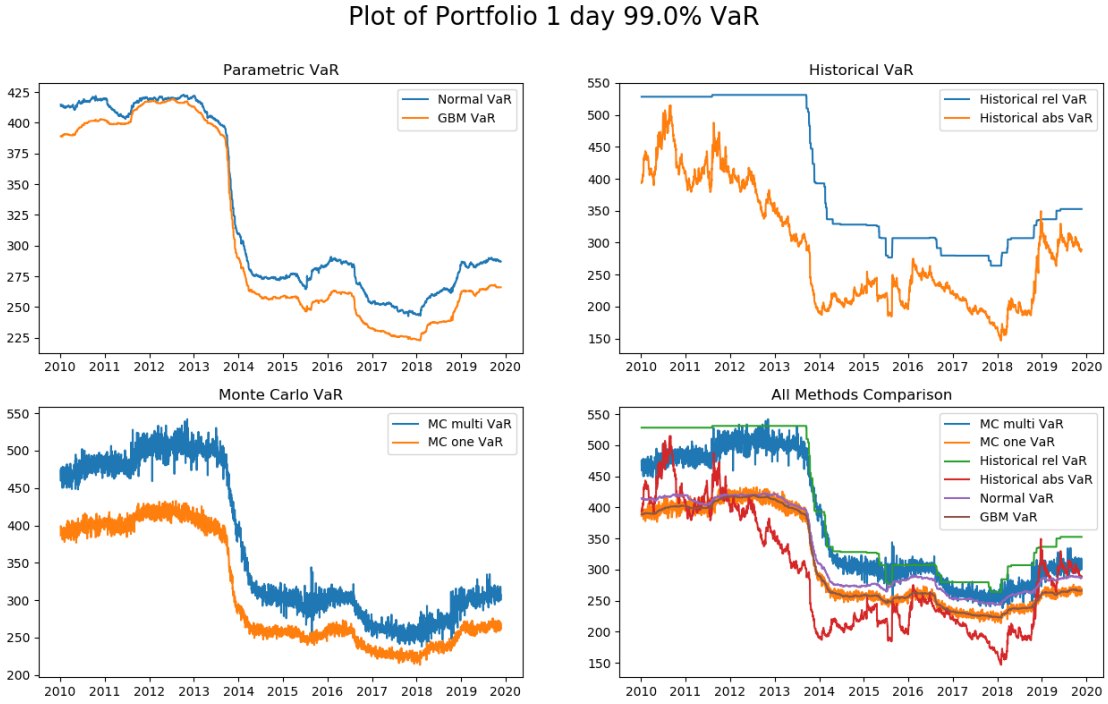
Plot of Portfolio 1 day 99.0% VaR
Parametric VaR
Normal VaR
GBM VaR
Historical VaR
Historical rel VaR
Historical abs VaR
Monte Carlo VaR
MC multi VaR
MC one VaR
All Methods Comparison
MC multi VaR
MC one VaR
Historical rel VaR
Historical abs VaR
Normal VaR
GBM VaR

Figure 5.38

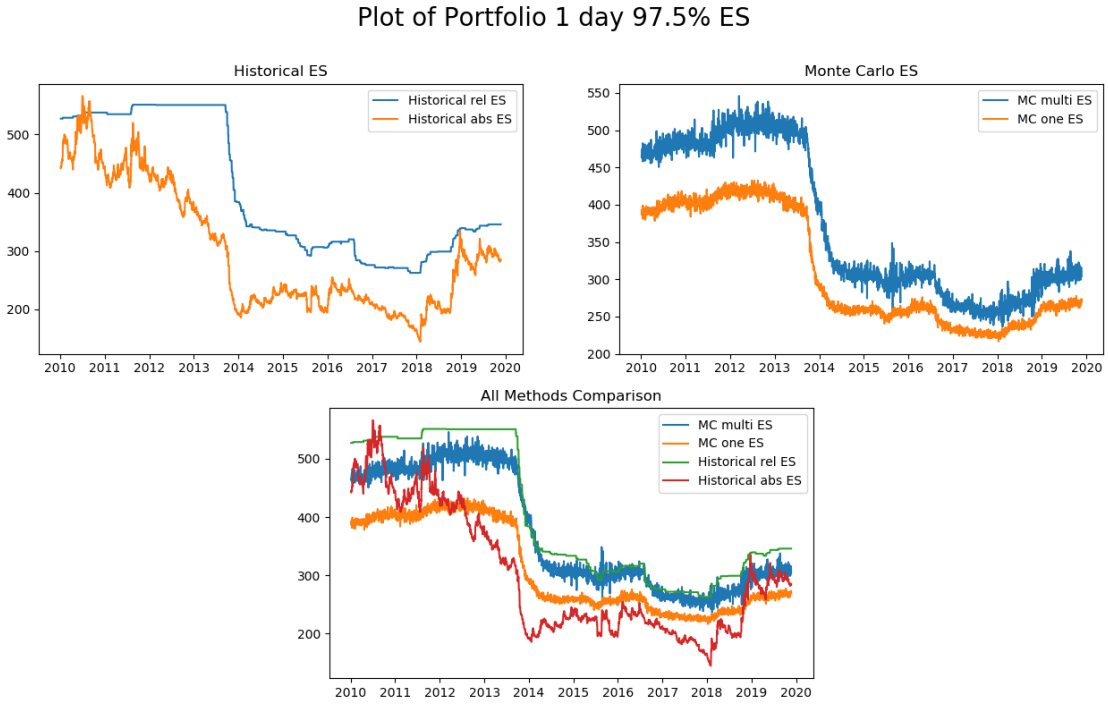
Plot of Portfolio 1 day 97.5% ES
Historical ES
Historical rel ES
Historical abs ES
Monte Carlo ES
MC multi ES
MC one ES
All Methods Comparison
MC multi ES
MC one ES
Historical rel ES
Historical abs ES

Figure 5.39

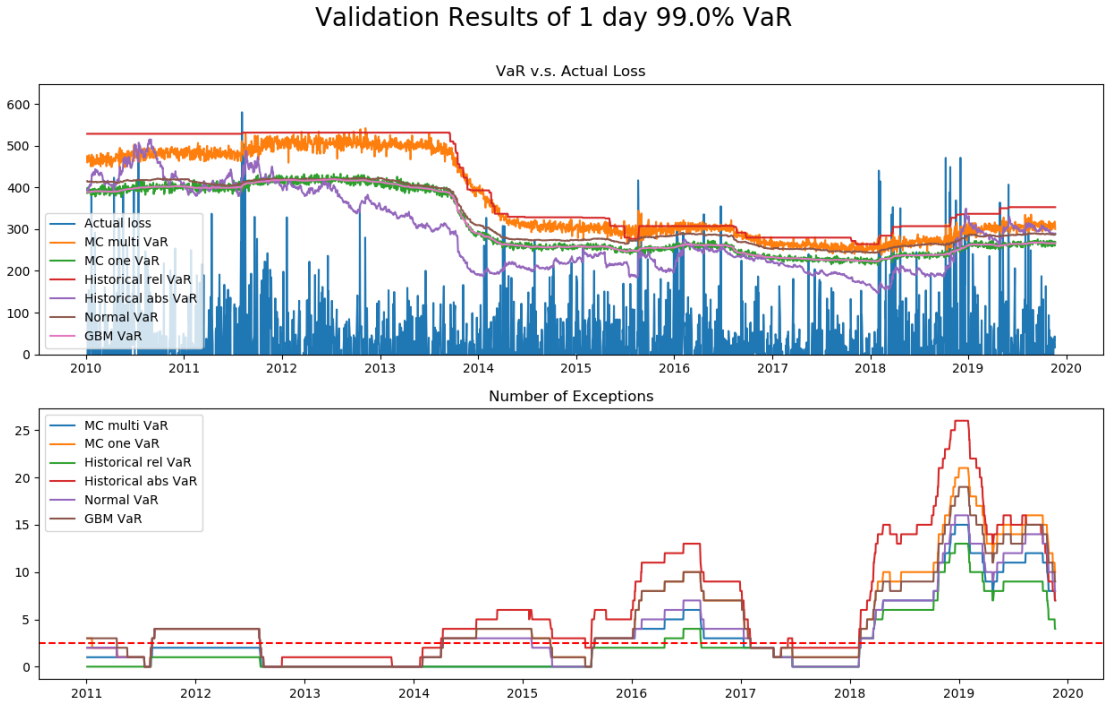

Figure 5.40

b. 5 days (1 week)

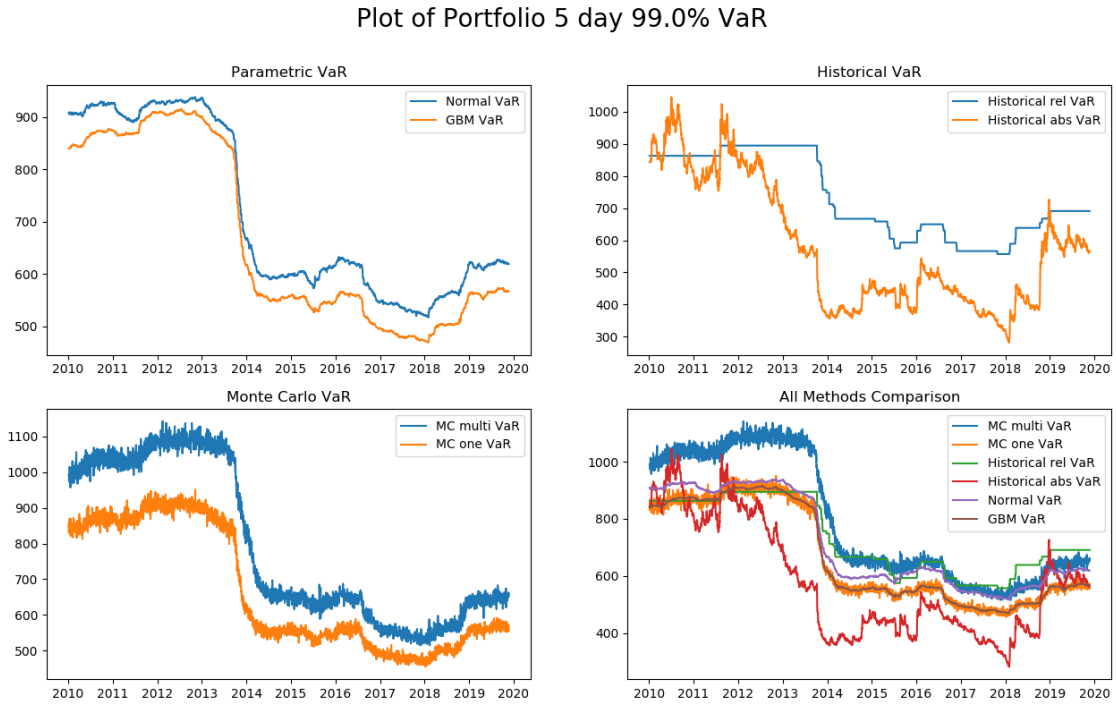
Plot of Portfolio 5 day 99.0% VaR
Parametric VaR
Normal VaR
GBM VaR
Historical VaR
Historical rel VaR
Historical abs VaR
Monte Carlo VaR
MC multi VaR
MC one VaR
All Methods Comparison
MC multi VaR
MC one VaR
Historical rel VaR
Historical abs VaR
Normal VaR
GBM VaR

Figure 5.41

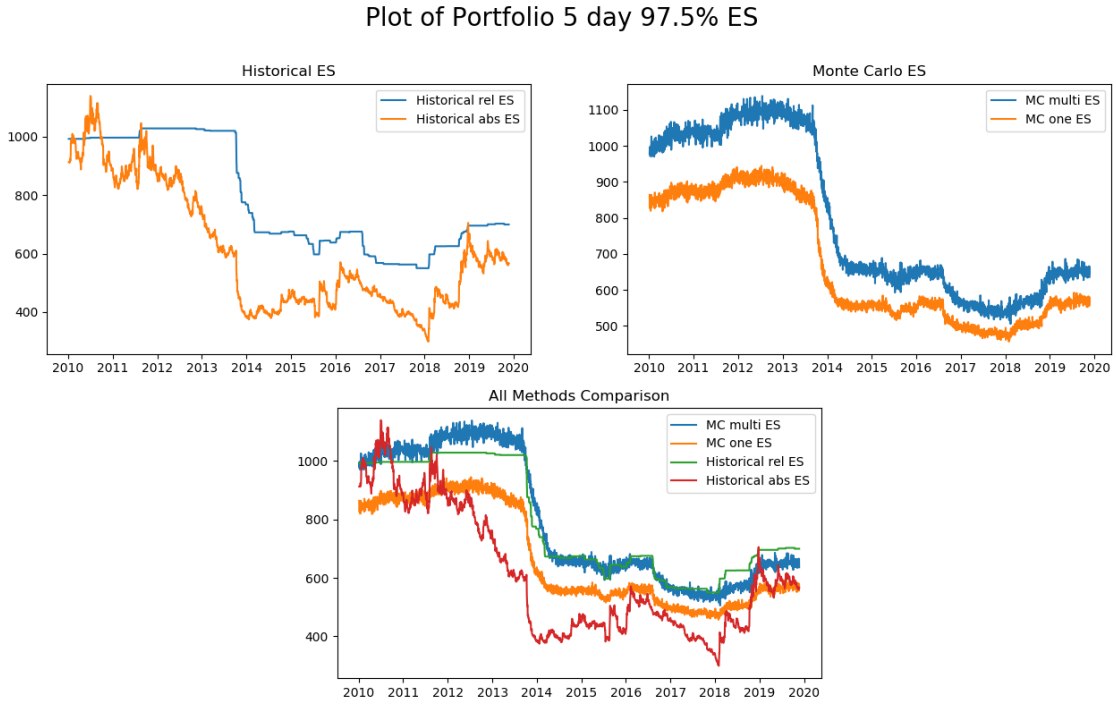
Plot of Portfolio 5 day 97.5% ES
Historical ES
Historical rel ES
Historical abs ES
Monte Carlo ES
MC multi ES
MC one ES
All Methods Comparison
MC multi ES
MC one ES
Historical rel ES
Historical abs ES

Figure 5.42

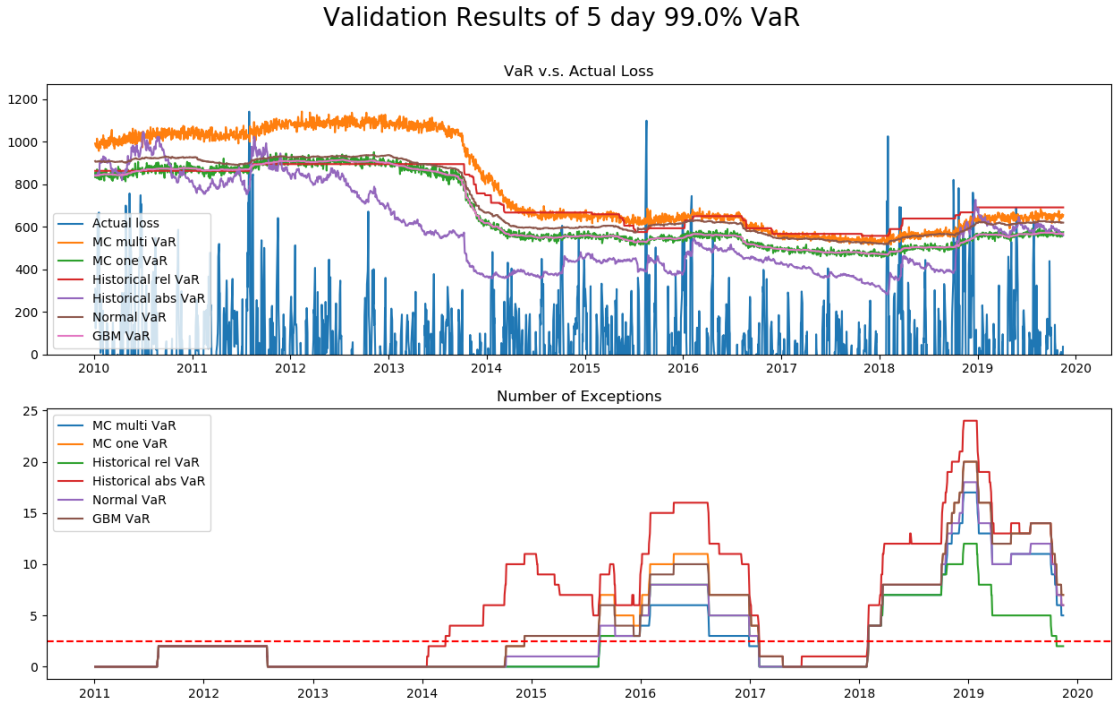

Figure 5.43

c. 20 days (1 month)

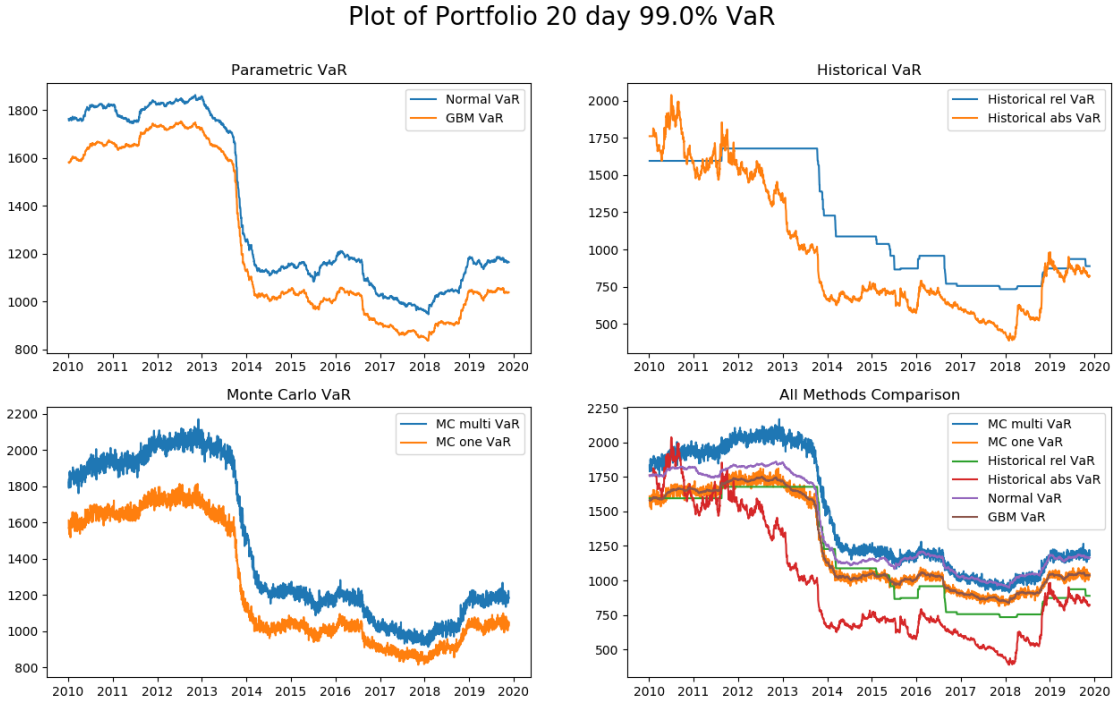
Plot of Portfolio 20 day 99.0% VaR
Parametric VaR
Normal VaR
GBM VaR
Historical VaR
Historical rel VaR
Historical abs VaR
Monte Carlo VaR
MC multi VaR
MC one VaR
All Methods Comparison
MC multi VaR
MC one VaR
Historical rel VaR
Historical abs VaR
Normal VaR
GBM VaR

Figure 5.44

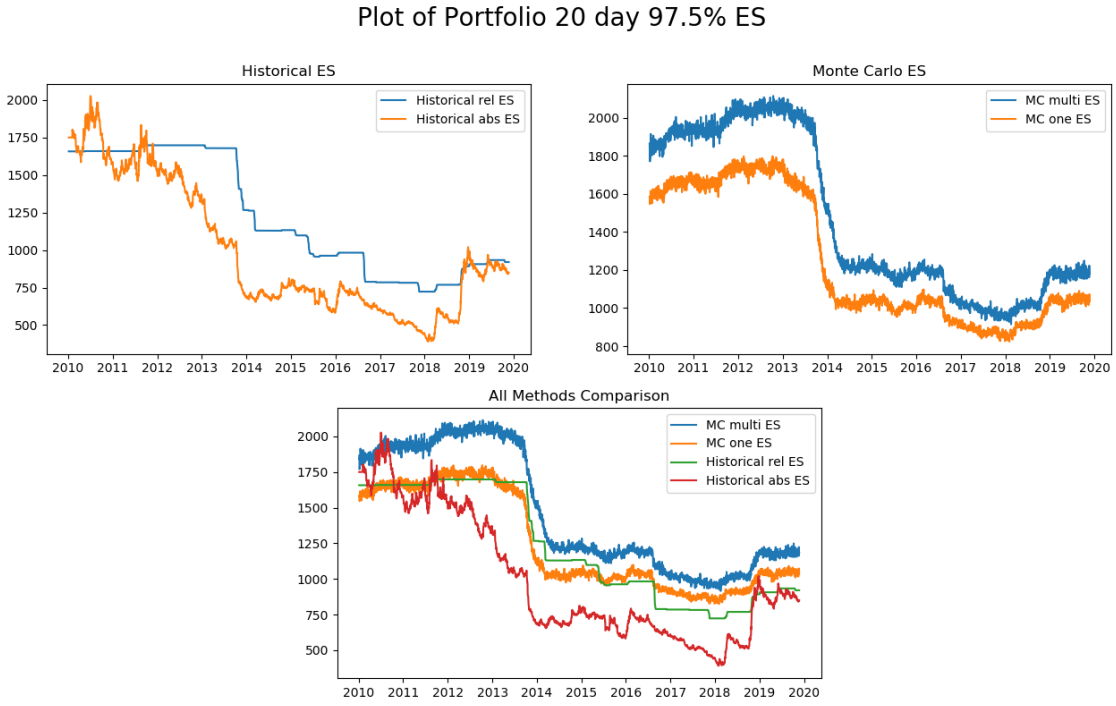
Plot of Portfolio 20 day 97.5% ES
Historical ES
Historical rel ES
Historical abs ES
Monte Carlo ES
MC multi ES
MC one ES
All Methods Comparison
MC multi ES
MC one ES
Historical rel ES
Historical abs ES

Figure 5.45

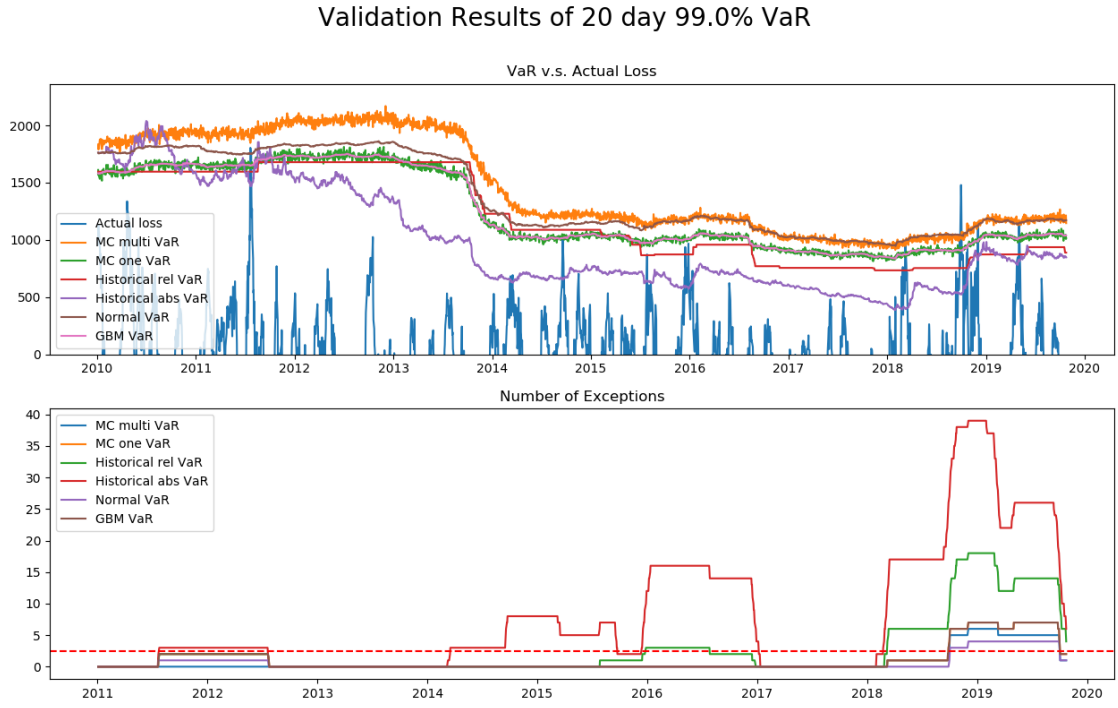

Figure 5.46

Similar to the results in changing VaR / ES and window length, VaR and ES movement has similar trends when horizon is set to 1 day, 1 week and 1 month. The number of exceptions is consistent except for historical VaR using absolute change method. This further validates that using relative change method in calculating historical VaR is better as we have discussed in section 3.3.1.

6 Conclusion

After thorough analysis of various risk measures and dedicated validation of model performance, we find that the system has the following strengths:

1. The system is capable of taking arbitrary combination of stocks and options with long or short position as input.

2. The system is user-friendly. Without the need to download stock data manually, users can acquire related data and calculation results by simply entering stock tickers as instructed.

3. Users can enter any parameters and the system will automatically convert them to feasible inputs. The system is able to automatically adjust the starting and ending time when stock data is available, and use it as the proper investment time period.

We also notice that the system has the following weakness:

1. Due to limited access to the Bloomberg terminal, it is not convenient for users to add new options to the current portfolio. Once they manually download option data, they can perform the same calculation on a portfolio with new options.

In general, we find that the weakness does not affect the solidity of our risk analysis. As a result, we can conclude that our approach is effective for risk calculation.

In addition, we have several interesting findings when backtesting the model:

1. Flipping the position of a portfolio, VaR and ES for short and long positions will have similar shapes. However, the short portfolio has a higher absolute value because the loss of a short position is unbounded.

2. Due to the correlations and hedging effect, the Parametric VaR assuming that the portfolio is normally distributed and the Monte Carlo Multi VaR/ ES will become larger in absolute value and be more volatile when the market moves together.

3. As the level of VaR/ ES decreases, the movement of VaR/ ES becomes less volatile.

4. Increasing the window length will make VaR and ES movement more stationary, given that we are analyzing risk measures in a longer period of time.

Finally, there is still room for improvement in our risk calculation system. If we would have access to the Bloomberg API, option data and the corresponding calculation results would be readily available by just entering the ticker of underlying assets in our system. This could further increase the efficiency of our system and creates an even more user-friendly interface environment.

References

[1] Xu, J., Razak, N. A. A., Robinson, C. W., & Jin, P. (n.d.). 2016-2017 CAE Actuarial Science Research Project. Retrieved December 17, 2019, from https://math.illinois.edu/system/files/inline-files/Proj8-AY1617-report.pdf.

[2] Dmouj, A. (n.d.). Theory and Practice - VU. Retrieved December 17, 2019, from https://beta.vu.nl/nl/Images/werkstuk-dmouj-tcm235-91341.pdf.

[3] Hull, J. (2009). Options, Futures, & Other Derivatives. Upper Saddle River, NJ: Prentice Hall.

[4] Strengths, Weaknesses, and Applications. (n.d.). Retrieved December 17, 2019, from https://valueatrisk.weebly.com/strengths-weaknesses-and-applications.html.

[5] Black–Scholes model. (n.d.). Retrieved December 17, 2019, from https://www.layscience.net/black-scholes-model/.

[6] Bank for International Settlements, BIS. Supervisory Framework for the use of 'Backtesting' in Conjunction with the Internal Models Approach to Market Risk Capital Requirements. https://www.bis.org/publ/bcbs22.pdf.

[7] QuantConnect Corporation. (n.d.). Retrieved December 18, 2019, from https://www.quantconnect.com/tutorials/introduction-to-options/options-pricing-black-scholes-merton-model.

[8] Kenton, W. (n.d.). How the Black Scholes Price Model Works. Retrieved December 18, 2019, from https://www.investopedia.com/terms/b/blackscholes.asp.

[9] Self-financing portfolio. (2019, December 17). Retrieved from https://en.wikipedia.org/wiki/Self-financing_portfolio.

[10] Björk Tomas. (2009). Arbitrage theory in continuous time (3rd ed.). Oxford: Oxford Univ. Press.

[11] Board of Governors of the Federal Reserve System. (n.d.). Retrieved December 19, 2019, from https://www.federalreserve.gov/releases/H15/default.htm.

A Appendix

A.1 Common Questions

My OS crashes after launching the software.

Unfortunately, if someone is using macOS 10.6 or later, the Apple-supplied Tcl/Tk 8.5 has serious bugs that can cause application crashes. To launch the software, do not use the Apple-supplied Pythons. Instead, install and use a newer version of Python from python.org or a third-party distributor that supplies or links with a newer version of Tcl/Tk.

See `https://www.python.org/download/mac/tcltk/#built-in-8-6-8` for more information.

The software crashes once launched.

You should check if all the required packages are installed correctly.

Typically, if the system reports 'No module named 'pandas_datareader', you should then install pandas_datareader by:

`>python -m pip install pandas_datareader`

Or in a JupyterNotebook, try:

`!pip install pandas_datareader`

Gernerally,

`>python -m pip install <package name>` will solve this type of problem.

For more installation helps please visit the official Python document: `https://docs.python.org/3/installing/index.html`

The system reports Error 'Input files not exist!'.

The system needs a txt file as the input of the portfolio. Make sure the 'Portfolio.txt' is correctly named and located in the same working directory with the software launcher.

The system reports Error 'Invalid inputs! Please check your Port-

folio.txt'.

The inputs you write down in 'Portfolio.txt' is not correct. Make sure the stocks are written as $< ticker >, < position >$ and options are written as $< ticker >, < position >, < type >, < maturity >$, one in each line. The position should be either 'call' or 'put'. The maturity should be chosen from '3', '6', '12'.

The system reports Error 'Options must have provided underlying stock!'.

To include options in a portfolio, the system requires option volatility data stored in the "Option_Data" folder. Make sure you have downloaded the complete folder and set it in the same working directory with the software launcher. Also, the option cannot have underlying outside the given 11 illustrations. They are AAPL, AMZN, BKNG, COF, CVS, DATE, GE, KO, NKE, NVDA, SPX, XRX.

The system reports Error 'Invalid inputs! Please check your setups'.

This error message is caused by invalid input parameters. For example, entering 99 instead of 0.99 for VaR Level. Please check all the input entries on calculation page.

The system reports Error 'WARNING: Portfolio value sign changes during the period, results may not be reliable!!!'.

This is caused by target portfolio value changing sign during the investment time period. The calculation results in this case are highly unreliable or completely trashed. You may take a look at unit portfolio value plot by clicking 'Portfolio' button, and consider modifying the positions

The system reports Error 'Window size / Horizon length cannot be longer than total data!'.

The input parameter 'Window Length' and 'Horizon period' should not be longer than the

total investment time period.

The system reports Error 'RemoteDataError'.

The ticker provided in 'Portfolio.txt' must be valid from Yahoo finance. Please check the spell of your tickers.

The system reports Error 'Please calculate result first'.

Please run the 'Calculation' button first before making any plots.

A.2 Software Update History

- Nov 29th, Version 0.5:

  A notebook contains all separate reading and calculating functions.

- Dec 7th, Version 0.6:

  Fixed errors from testing functions for various input parameters and data.

- Dec 9th, Version 0.8:

  Integrated major reading and calculating functions.

  Collected separate output figures into subplot combinations.

  Wrote all utility functions into one Python script.

- Dec 10th, Version 1.0:

  Developed a GUI prototype for integrated functions in notebook.

  Use Tkinter for page frames and control panels.

- Dec 11th, Version 1.1:

  Updated GUI design and commands.

  Tested and added in-app handlers for various error cases.

- Dec 11th, Version 1.2:

  Added and tested save functions.

  Converted the GUI notebook to a Python script.

- Dec 13th, Version 1.3:

  Fixed incompatible issues for switching operating systems.

  Updated links to software help documents.

A.3 Codes for Utility Functions

```
# python3
# -*- coding: utf-8 -*-
"""
Created on Wed Dec 11 02:15:50 2019

@author: yz3380@columbia.edu
"""

import os
import glob
import operator
import itertools
import numpy as np
import pandas as pd
import scipy.stats as st
import scipy.optimize as so
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
from datetime import datetime
from pandas_datareader import data as web
from pandas_datareader._utils import RemoteDataError
from matplotlib.lines import Line2D

# check inputs
def Check_input(Time_start, Time_end, VaR_p, ES_p, Initial_value, Window, Horizon_period, Parameter):
    assert(isinstance(Time_start, pd.Timestamp))
    assert(isinstance(Time_end, pd.Timestamp))
    assert(Time_start < Time_end)
    assert(isinstance(VaR_p, (int, float)))
    assert(isinstance(ES_p, (int, float)))
    assert(0 < VaR_p < 1)
    assert(0 < ES_p < 1)
    assert(isinstance(Initial_value, (int, float)))
    assert(Initial_value > 0)
    assert(isinstance(Window, (int, float)))
    assert(Window > 0)
    assert(isinstance(Horizon_period, int))
    assert(Horizon_period > 0)
    assert(Parameter in ('Window', 'Exponential'))
    return

# read underlyings and options data from current working directory
# can read unlimited number of files, as long as correct CSV format is provided
def Read_option(path):
    try:
        all_files = glob.glob(os.path.join(path, 'Option_Data', '*.csv'))

        option_df = {}
        for file in all_files:
            stock_name = pd.read_csv(file).columns[0].upper() + '_option'
            df = pd.read_csv(file, skiprows=1)
            df.index = pd.to_datetime(df.iloc[:, 0])
            df = df.iloc[:, 1:]
            df = df.fillna(method='ffill')
            df.iloc[:, 1:] = df.iloc[:, 1:] / 100
            option_df[stock_name] = df

    except FileNotFoundError:
        raise FileNotFoundError('Invalid␣directory␣or␣file␣name!')
```

```
    else:
        return option_df

# read portfolio positions
def Read_position(path, option_df):
    try:
        portfolio = os.path.join(path, 'Portfolio.txt')
        stocks = {}
        options = {}
        with open(portfolio) as f:
            for line in f.readlines():
                if line[0] in ['#', '-', '\n']:
                    continue
                else:
                    line = line.strip()
                    line = line.replace(' ', '')
                    content = line.split(',')
                    if len(content) == 2:                                    # read stock
                        assert(isinstance(content[0], str))
                        assert(isinstance(float(content[1]), float))
                        stocks[content[0].upper()] = float(content[1])
                    elif len(content) == 4:                                  # read option
                        assert(isinstance(content[0], str))
                        assert(isinstance(float(content[1]), float))
                        assert(content[2].lower() in ('call', 'put'))
                        assert(int(content[3]) in (3, 6, 12))
                        name = content[0].upper() + '_option'
                        options[name] = (float(content[1]), content[2].lower(), int(content[3]))
                        if not name in option_df.keys():
                            raise NameError
                    else:
                        raise ValueError
            assert(len(stocks) + len(options) > 0)

    except (AssertionError, ValueError):
        raise ValueError('Invalid inputs! Please check your Portfolio.txt')
    except FileNotFoundError:
        raise FileNotFoundError('Input files not exist!')
    except NameError:
        raise NameError('Options must have provided underlying stocks!')
    else:
        return stocks, options

# adjust variables according to portfolio positions
def Adjust_option_time(option_df, options, start, end):
    opt_start = option_df[next(iter(option_df))].index[0]
    opt_end = option_df[next(iter(option_df))].index[-1]

    if bool(options):                                                        # portfolio include options, adjust time period
        if start < opt_start:
            start = opt_start
        if end > opt_end:
            end = opt_end

    return start, end

# download stocks price from Yahoo
def Read_stock(stocks, start, end):
    stock_data = {}
    adj_start = start
    adj_end = end
```

```
    if adj_start < pd.Timestamp('1980-01-01'):                                  # set lower bound for starting period
        adj_start = pd.Timestamp('1980-01-01')
    try:
        for ticker in stocks.keys():
            df = pd.DataFrame()
            df[ticker] = web.DataReader(ticker, 'yahoo', adj_start, adj_end)['Close']
            df = df.fillna(method='ffill')
            stock_data[ticker] = df

            if adj_start < stock_data[ticker].index[0]:
                adj_start = stock_data[ticker].index[0]
            if adj_end > stock_data[ticker].index[-1]:
                adj_end = stock_data[ticker].index[-1]

        for ticker in stock_data.keys():                                        # adjust dataframe to same length
            stock_data[ticker] = stock_data[ticker].truncate(before=adj_start, after=adj_end)

        if len(stock_data) > 0:
            combine_stock = pd.concat([stock_data[x] for x in stock_data.keys()], axis = 1)
            combine_stock = combine_stock.fillna(method='ffill')
            for ticker in combine_stock.columns:
                stock_data[ticker] = combine_stock[ticker]

    except OverflowError as err:
        raise OverflowError(err)
    except RemoteDataError as err:
        raise RemoteDataError(err)
    except KeyError as err:
        raise KeyError(err)
    else:
        return stock_data, adj_start, adj_end

# load 1 year Treasury bill rate
def Read_interest(path, start, end):
    try:
        FRB = os.path.join(path, 'Option_Data', 'FRB_H15.csv')

        df = pd.read_csv(FRB)
        df.index = pd.to_datetime(df.iloc[:, 0])
        df = df.iloc[:, 1]
        df = df.fillna(method='ffill')
        df = df.truncate(before=start, after=end)
        df= df / 100

    except FileNotFoundError:
        raise FileNotFoundError('(Interest␣rate)Invalid␣directory␣or␣file␣name!')
    else:
        return df

# Construct portfolio value and 1 day log return data, also truncated option data frame
def Portfolio_return(stock_df, option_df, int_rate, stocks, options, start, end):
    for ticker in option_df.keys():
        option_df[ticker] = option_df[ticker].truncate(before=start, after=end)  # truncate option data to adjusted time period

    if len(stock_df) > 0:
        index = stock_df[next(iter(stock_df))].index
    else:
        index = option_df[next(iter(option_df))].index
    N = len(index)

    portfolio = np.zeros(N)
```

```
    for ticker in stocks:
        portfolio += stock_df[ticker].values * stocks[ticker] #
    for ticker in options:
        iv_index = 3 if options[ticker][1] == 'call' else 0                    # set index for option implied volatility
        iv_lib = dict(zip([3, 6, 12], [1, 2, 3]))
        iv_index = iv_index + iv_lib[options[ticker][2]]
        portfolio += Black_Scholes(option_df[ticker].iloc[:, 0].values, option_df[ticker].iloc[:, 0].values, int_rate.values,\
                                   option_df[ticker].iloc[:,iv_index].values, options[ticker][2]/12, options[ticker][1])\
                     * options[ticker][0]

    if portfolio[0] > 0:                                                       # set long / short state
        long_portfolio = True
        portfolio = portfolio.clip(min=0.01)                                   # avoid negative value in long portfolio
    else:
        long_portfolio = False
        portfolio = portfolio.clip(max=-0.01)

    portfolio_past = portfolio[:-1]                                            # calculate one day log return
    portfolio = portfolio[1:]
    log_return =np.log(portfolio / portfolio_past)
    p_return = pd.DataFrame({'Price':portfolio})
    p_return['Return'] = log_return
    p_return.set_index(index[1:], inplace = True)
    p_return.fillna(method='ffill')

    for ticker in stock_df.keys():                                             # update the stock data
        price_past = stock_df[ticker].values[:-1]
        price = stock_df[ticker].values[1:]
        log_return = np.log(price / price_past)
        s_return = pd.DataFrame({'Price':price})
        s_return['Return'] = log_return
        s_return.set_index(index[1:], inplace = True)
        stock_df[ticker] = s_return
    for ticker in options.keys():                                              # add option underlying to stock data
        name = ticker.replace('_option', '')
        if not name in stock_df.keys():
            price_past = option_df[ticker].iloc[:, 0].values[:-1]
            price = option_df[ticker].iloc[:, 0].values[1:]
            log_return = np.log(price / price_past)
            s_return = pd.DataFrame({'Price':price})
            s_return['Return'] = log_return
            s_return.set_index(index[1:], inplace = True)
            stock_df[name] = s_return

    return p_return, option_df, stock_df, long_portfolio

# construct relative change data frame
def Historical_change(port_df, period):
    index = port_df.index
    portfolio = port_df.iloc[:, 0].values

    if port_df.shape[0] < period:
        raise ValueError('Horizon␣length␣cannot␣be␣longer␣than␣total␣data!')

    portfolio_past = portfolio[:-period]                                       # calculate k day difference
    portfolio = portfolio[period:]
    rel_return = portfolio / portfolio_past
    abs_return = portfolio - portfolio_past

    p_return = pd.DataFrame({'Price':portfolio})
    p_return['Relative'] = rel_return
```

```
    p_return['Absolute'] = abs_return
    p_return.set_index(index[period:], inplace = True)

    return p_return

# calibrate GBM parameter, window version
def Calibrate(df, l, period):
    T = round(l * 252)
    N = df.shape[0]
    if N <= T + period:
        raise ValueError('Window size cannot be longer than total data!')

    drift = []
    sigma = []
    for start in range(N - T + 1):
        end = start + T
        mu = sum(df.iloc[start:end, 1]) / T
        var = sum(df.iloc[start:end, 1] ** 2) / T - mu ** 2
        sigma.append(np.sqrt(var) * np.sqrt(252))
        drift.append(mu * 252 + sigma[-1] ** 2 / 2)

    data = {'drift':drift, 'sigma':sigma}
    parameter = pd.DataFrame(data, index = df.index[T - 1:])
    return parameter

# discrepancy function in estimating lambda
def Estimate_lambda(_lambda, l):
    N = 252 * l
    Distance = (2 *_lambda ** (N + 1) + 2 * _lambda ** N - _lambda * (N + 1) + N - 1) / ((_lambda + 1) * N)
    return Distance

# calibrate GBM parameter, equivalent exponential version
def Exp_Calibrate(df, l, period):
    T = round(l * 252)
    N = df.shape[0]
    if N <= T + period:
        raise ValueError('Window size cannot be longer than total data!')

    optimize = so.minimize(Estimate_lambda, x0=1, args=l)                        # calculate lambda
    _lambda = float(optimize.x)
    drift = []
    sigma = []

    for start in range(N - T + 1):
        end = start + T
        weight = np.array(list(itertools.accumulate([_lambda] * (end),
                                                    operator.mul))) * (1 - _lambda) / _lambda
        weight = np.fliplr([weight])[0]
        #weight = [_lambda ** (end - i - 1) * (1 - _lambda) for i in range(end)] # loop version, slow in computation
        mu = sum(df.iloc[:end, 1] * weight)
        var = sum(df.iloc[:end, 1] ** 2 * weight) - mu ** 2
        sigma.append(np.sqrt(var) * np.sqrt(252))
        drift.append(mu * 252 + sigma[-1] ** 2 / 2)

    data = {'drift':drift, 'sigma':sigma}
    result = pd.DataFrame(data, index = df.index[T - 1:])
    return result

# calibrate window GBM parameters for all stocks
def Calibrate_all(stock_df, l, period):
    gbm_stocks = {}
```

```
    for ticker in stock_df.keys():
        gbm_stocks[ticker] = Calibrate(stock_df[ticker], l, period)
    return gbm_stocks

# calibrate exp GBM parameters for all stocks
def Calibrate_all_EW(stock_df, l, period):
    gbm_stocks_ew = {}
    for ticker in stock_df.keys():
        gbm_stocks_ew[ticker] = Exp_Calibrate(stock_df[ticker], l, period)
    return gbm_stocks_ew

# calculate correlation
def Calculate_rho(stock_df, l):
    T = round(l * 252)
    N = stock_df[next(iter(stock_df))].shape[0]
    combine_df = pd.DataFrame()
    for ticker in stock_df.keys():
        combine_df[ticker] = stock_df[ticker]['Return']

    rho = []
    for start in range(N - T + 1):
        end = start + T
        df = combine_df.iloc[start:end, :]
        rho.append(df.corr())

    return rho

# Black Scholes option pricing, array version
def Black_Scholes(S_0, K, r, sigma, T, opt_type):                    # S_0, K, r, sigma are arrays
    d1 = (np.log(S_0 / K) + (r + 0.5 * sigma ** 2) * T) / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)
    c_price = st.norm.cdf(d1) * S_0 - K * np.exp(-r * T) * st.norm.cdf(d2)
    p_price = K * np.exp(-r * T) * st.norm.cdf(-d2) - st.norm.cdf(-d1) * S_0

    if opt_type == 'call':
        return c_price
    else:
        return p_price

# long and short VaR, array version
def Long_VaR(V, mu, sigma, period, level):                           # mu, sigma array
    VaR = V - V * np.exp((mu - 0.5 * sigma ** 2) * period + sigma * np.sqrt(period) * st.norm.ppf(1 - level))
    return VaR

def Short_VaR(V, mu, sigma, period, level):
    VaR = V * np.exp((mu - 0.5 * sigma ** 2) * period + sigma * np.sqrt(period) * st.norm.ppf(level)) - V
    return VaR

# parametric GBM VaR
def Parametric_VaR_GBM(parameters, V, period, level, state):
    period = period / 252
    index = parameters.index
    mu = parameters.iloc[:, 0].values
    sigma = parameters.iloc[:, 1].values

    if state == True:                                                # long portfolio
        VaR = Long_VaR(V, mu, sigma, period, level)
    else:                                                            # short portfolio
        VaR = Short_VaR(V, mu, sigma, period, level)

    result = pd.DataFrame({'GBM␣VaR':VaR}, index = index)
```

```
    return result

# parametric VaR, approximate Normal, STOCKS ONLY
def Parametric_VaR_Normal(stock_df, stock_parameters, stocks, rho, V, l, period, level, state):
    period = period / 252
    index = stock_parameters[next(iter(stock_parameters))].index
    N = len(index)
    T = round(l * 252)
    portfolio_value = np.sum([stock_df[x].iloc[T - 1:, 0].values for x in stocks.keys()])

    ev = np.zeros(N)
    ev2 = np.zeros(N)
    portfolio_value = np.zeros(N)
    for ticker1 in stocks.keys():                                          # calculate normal approximate parameters
        portfolio_value += stock_df[ticker1].iloc[T - 1:, 0].values * stocks[ticker1]
        a = stocks[ticker1] * stock_df[ticker1].iloc[T - 1:, 0].values
        mu1 = stock_parameters[ticker1].iloc[:, 0].values * period
        sigma1 = stock_parameters[ticker1].iloc[:, 1].values * np.sqrt(period)
        ev += a * np.exp(mu1)
        ev2 += a ** 2 * np.exp(2 * mu1 + sigma1 ** 2)
        for ticker2 in stocks.keys():
            if ticker2 != ticker1:
                b = stocks[ticker2] * stock_df[ticker2].iloc[T - 1:, 0].values
                mu2 = stock_parameters[ticker2].iloc[:, 0].values * period
                sigma2 = stock_parameters[ticker2].iloc[:, 1].values * np.sqrt(period)

                rho_ab = np.array([x[ticker1][ticker2] for x in rho])      # get correlation array
                ev2 += a * b * np.exp(mu1 + mu2 + rho_ab * sigma1 * sigma2)

    ev2 = ev2 - ev ** 2
    sd = np.sqrt(ev2)

    if state == True:                                                      # long portfolio
        VaR = V - (ev + sd * st.norm.ppf(1 - level)) * V / portfolio_value
    else:                                                                  # short portfolio
        VaR = - (ev + sd * st.norm.ppf(level)) * V / portfolio_value + V
    result = pd.DataFrame({'Normal␣VaR':VaR}, index = index)
    return result

# Calculate historical relative and absolute VaR
def Historical_VaR(his_df, V, l, level, state):
    T = round(l * 252)
    N = his_df.shape[0]
    rel_VaR = []
    abs_VaR = []
    for start in range(N - T + 1):
        end = start + T
        if state == True:
            relative = np.quantile(his_df.iloc[start:end, 1], 1 - level, interpolation='lower')
            absolute = np.quantile(his_df.iloc[start:end, 2], 1 - level, interpolation='lower')
            rel_VaR.append(V - V * relative)
            abs_VaR.append(-absolute * V / his_df.iloc[end - 1, 0])
        else:
            relative = np.quantile(his_df.iloc[start:end, 1], level, interpolation='higher')
            absolute = np.quantile(his_df.iloc[start:end, 2], level, interpolation='higher')
            rel_VaR.append(V * relative - V)
            abs_VaR.append(-absolute * V / his_df.iloc[end - 1, 0])

    result = pd.DataFrame({'Historical␣rel␣VaR':rel_VaR, 'Historical␣abs␣VaR':abs_VaR}, index = his_df.index[T - 1:])
    return result
```

```
# calculate MC VaR as one portfolio
def MC_VaR_One(p_parameters, V, period, level, sample, state):
    period = period / 252
    N = p_parameters.shape[0]

    VaR = []
    for i in range(N):
        mu = p_parameters.iloc[i, 0] * period
        sigma = p_parameters.iloc[i, 1] * np.sqrt(period)
        r = np.random.normal(0, sigma, sample)
        if state == True:
            log_r = np.quantile(r, 1 - level)
            VaR.append(V - V * np.exp(mu - 0.5 * sigma ** 2 + log_r))
        else:
            log_r = np.quantile(r, level)
            VaR.append(V * np.exp(mu - 0.5 * sigma ** 2 + log_r) - V)

    result = pd.DataFrame({'MC one VaR':VaR}, index = p_parameters.index)
    return result

# calculate MC VaR by simulating multiple stocks
def MC_VaR_Multi(s_parameters, port_df, option_df, stock_df, stocks, options,
                 rho, int_rate, V, l, period, level, sample, state):
    period = period / 252
    T = round(l * 252)
    index = s_parameters[next(iter(s_parameters))].index
    N = len(index)

    VaR = []
    for i in range(N):
        mu = np.array([0 for x in stock_df.keys()])
        sigma = np.array([s_parameters[x].iloc[i, 1] for x in stock_df.keys()])
        cov = (np.diag(sigma) @ np.diag(sigma).T) @ rho[i].values

        r = pd.DataFrame(np.random.multivariate_normal(mu, cov, sample, tol=1))  # stock price using multi variate normal
        r.columns = [ticker for ticker in stock_df.keys()]

        portfolio = np.zeros(sample)
        for ticker in stocks:
            price = np.exp((s_parameters[ticker].iloc[i, 0] - 0.5 * s_parameters[ticker].iloc[i, 1] ** 2) * period
                           + np.sqrt(period) * r[ticker].values) * stock_df[ticker].iloc[T-1+i, 0]
            portfolio += price * stocks[ticker]
        for ticker in options.keys():
            name = ticker.replace('_option', '')
            iv_index = 3 if options[ticker][1] == 'call' else 0                  # set index for option implied volatility
            iv_lib = dict(zip([3, 6, 12], [1, 2, 3]))
            iv_index = iv_index + iv_lib[options[ticker][2]]

            price = np.exp((s_parameters[name].iloc[i, 0] - 0.5 * s_parameters[name].iloc[i, 1] ** 2) * period
                           + np.sqrt(period) * r[name].values) * stock_df[name].iloc[T-1+i, 0]
            portfolio += Black_Scholes(price, price, int_rate.values[T-1+i], option_df[ticker].iloc[T-1+i,iv_index], \
                                       options[ticker][2]/12, options[ticker][1]) * options[ticker][0]

        portfolio = portfolio * V / port_df.iloc[T-1+i, 0]

        if state == True:
            VaR_temp = np.quantile(portfolio, 1 - level, interpolation = 'lower')
            VaR.append(V - VaR_temp)
        else:
            VaR_temp = np.quantile(portfolio, level, interpolation = 'higher')
            VaR.append(VaR_temp - V)
```

```
    result = pd.DataFrame({'MC multi VaR':VaR}, index = index)
    return result

# Calculate historical relative and absolute ES
def Historical_ES(his_df, V, l, level, state):
    T = round(l * 252)
    N = his_df.shape[0]
    rel_ES = []
    abs_ES = []
    for start in range(N - T + 1):
        end = start + T
        if state == True:
            relative = np.quantile(his_df.iloc[start:end, 1], 1 - level, interpolation='lower')
            absolute = np.quantile(his_df.iloc[start:end, 2], 1 - level, interpolation='lower')

            value1 = V * his_df.iloc[start:end, 1].values
            value1 = value1[value1 < relative * V]
            rel_ES.append(V - np.mean(value1))

            value2 = -his_df.iloc[start:end, 2].values * V / his_df.iloc[end - 1, 0]
            value2 = value2[value2 > -absolute * V / his_df.iloc[end - 1, 0]]
            abs_ES.append(np.mean(value2))
        else:
            relative = np.quantile(his_df.iloc[start:end, 1], level, interpolation='higher')
            absolute = np.quantile(his_df.iloc[start:end, 2], level, interpolation='higher')

            value1 = V * his_df.iloc[start:end, 1].values
            value1 = value1[value1 > relative * V]
            rel_ES.append(np.mean(value1) - V)

            value2 = - his_df.iloc[start:end, 2].values * V / his_df.iloc[end - 1, 0]
            value2 = value2[value2 > - absolute * V / his_df.iloc[end - 1, 0]]
            abs_ES.append(np.mean(value2))

    result = pd.DataFrame({'Historical rel ES':rel_ES, 'Historical abs ES':abs_ES},
                          index = his_df.index[T - 1:])
    return result

# calculate MC ES as one portfolio
def MC_ES_One(p_parameters, V, period, level, sample, state):
    period = period / 252
    N = p_parameters.shape[0]

    ES = []
    for i in range(N):
        mu = p_parameters.iloc[i, 0] * period
        sigma = p_parameters.iloc[i, 1] * np.sqrt(period)
        r = np.random.normal(0, sigma, sample)
        if state == True:
            log_r = np.quantile(r, 1 - level)
            VaR = V - V * np.exp(mu - 0.5 * sigma ** 2 + log_r)
            value1 = V - V * np.exp(mu - 0.5 * sigma ** 2 + r)
            value1 = value1[value1 > VaR]
            ES.append(np.mean(value1))
        else:
            log_r = np.quantile(r, level)
            VaR = V * np.exp(mu - 0.5 * sigma ** 2 + log_r) - V
            value2 = V * np.exp(mu - 0.5 * sigma ** 2 + r) - V
            value2 = value2[value2 > VaR]
            ES.append(np.mean(value2))
```

```
    result = pd.DataFrame({'MC one ES':ES}, index = p_parameters.index)
    return result

# calculate MC ES by simulating multiple stocks
def MC_ES_Multi(s_parameters, port_df, option_df, stock_df, stocks, options,
                rho, int_rate, V, l, period, level, sample, state):
    period = period / 252
    T = round(l * 252)
    index = s_parameters[next(iter(s_parameters))].index
    N = len(index)

    ES = []
    for i in range(N):
        mu = np.array([0 for x in stock_df.keys()])
        sigma = np.array([s_parameters[x].iloc[i, 1] for x in stock_df.keys()])
        cov = (np.diag(sigma) @ np.diag(sigma).T) @ rho[i].values

        r = pd.DataFrame(np.random.multivariate_normal(mu, cov, sample, tol=1))  # stock price using multi variate normal
        r.columns = [ticker for ticker in stock_df.keys()]

        portfolio = np.zeros(sample)
        for ticker in stocks:
            price = np.exp((s_parameters[ticker].iloc[i, 0] - 0.5 * s_parameters[ticker].iloc[i, 1] ** 2) * period
                           + np.sqrt(period) * r[ticker].values) * stock_df[ticker].iloc[T-1+i, 0]
            portfolio += price * stocks[ticker]
        for ticker in options.keys():
            name = ticker.replace('_option', '')
            iv_index = 3 if options[ticker][1] == 'call' else 0                  # set index for option implied volatility
            iv_lib = dict(zip([3, 6, 12], [1, 2, 3]))
            iv_index = iv_index + iv_lib[options[ticker][2]]

            price = np.exp((s_parameters[name].iloc[i, 0] - 0.5 * s_parameters[name].iloc[i, 1] ** 2) * period
                           + np.sqrt(period) * r[name].values) * stock_df[name].iloc[T-1+i, 0]
            portfolio += Black_Scholes(price, price, int_rate.values[T-1+i], option_df[ticker].iloc[T-1+i,iv_index], \
                                       options[ticker][2]/12, options[ticker][1]) * options[ticker][0]

        portfolio = portfolio * V / port_df.iloc[T-1+i, 0]

        if state == True:
            VaR = V - np.quantile(portfolio, 1 - level, interpolation = 'lower')
            value1 = V - portfolio
            value1 = value1[value1 > VaR]
            ES.append(np.mean(value1))
        else:
            VaR = np.quantile(portfolio, level, interpolation = 'higher') - V
            value2 = portfolio - V
            value2 = value2[value2 > VaR]
            ES.append(np.mean(value2))

    result = pd.DataFrame({'MC multi ES':ES}, index = index)
    return result

# Backtest function, parametric and MC ONLY
def Backtesting(port_df, VaR_df, V, period, state):
    N = VaR_df.shape[0]
    n = 252
    v_start = VaR_df.index[0]
    #v_end = VaR_df.index[-1]
    p_start = port_df.index.get_loc(v_start)
    #p_end = port_df.index.get_loc(v_end)
```

```
    exception = []
    for i in range(N - n - period):
        VaR = np.array(VaR_df.iloc[i:i + n, 0])
        price_change = port_df['Price'].values[p_start + i + period:p_start + i + period + n] \
        - port_df['Price'].values[p_start + i:p_start + i + n]
        Price_change = np.array(price_change) / np.array(port_df['Price'][p_start + i:p_start + i + n]) * V

        if state == True:
            count = -Price_change > VaR
            count = np.sum(count)
            exception.append(count)
        else:
            count = Price_change > VaR
            count = np.sum(count)
            exception.append(count)

    result = pd.DataFrame({'Exception':exception}, index = VaR_df.index[n - 1:N - period - 1])
    return result

# Stock Info Button: plot portfolio info
def Plot_Position(port_df, stock_df, option_df, stocks, options, int_rate, gbm_df):

    fig, ax = plt.subplots(2, 2, figsize=(15, 10))
    df = pd.DataFrame()
    if isinstance(stock_df, pd.core.frame.DataFrame):
        df['Portfolio'] = stock_df['Price']
    else:
        for ticker in stocks.keys():
            df[ticker] = stock_df[ticker]['Price']

    name = []
    value = []

    for ticker in stocks:
        name.append(ticker + ': ' + str(stocks[ticker]))
        value.append(abs(stock_df[ticker].iloc[0, 0] * stocks[ticker]))
    for ticker in options:
        name.append(ticker + ': '+ str(options[ticker][0]))
        iv_index = 3 if options[ticker][1] == 'call' else 0                    # set index for option implied volatility
        iv_lib = dict(zip([3, 6, 12], [1, 2, 3]))
        iv_index = iv_index + iv_lib[options[ticker][2]]
        value.append(abs(float(Black_Scholes(option_df[ticker].iloc[0, 0],
                                             option_df[ticker].iloc[0, 0], int_rate.values[0],
                                             option_df[ticker].iloc[0,iv_index],
                                             options[ticker][2]/12,
                                             options[ticker][1]) * options[ticker][0])))
    explode = [0.1] * len(name)

    fig.suptitle('Visualization of Portfolio Info', fontsize=20)
    ax[0, 0].pie(value, explode=explode, labels=name, shadow=True)
    ax[0, 0].axis('equal')
    ax[0, 0].legend(name, title="Assets", loc="best")
    ax[0, 0].set_title("Initial Asset Allocation")
    if bool(stocks):
        ax[0, 1].plot(df)
        ax[0, 1].set_title('Historical Stock Prices')
        ax[0, 1].legend(df.columns, loc='best')
    ax[1, 0].plot(port_df['Price'])
    ax[1, 0].set_title('Unit Portfolio Value')
    ax[1, 1].plot(gbm_df)
```

```
    ax[1, 1].set_title('Portfolio return drift and volatility')
    ax[1, 1].legend(gbm_df.columns, loc='best')
    plt.show()
    #return fig


# Plot VaR Button, 3 different methods VaR and comparison
def Plot_VaR(df, stocks, period, vlevel):
    fig, ax = plt.subplots(2, 2, figsize=(15, 10))

    fig.suptitle('Plot of Portfolio {} day {:.1%} VaR'.format(period, vlevel), fontsize=20)
    if bool(stocks):
        ax[0, 0].plot(df[['Normal VaR', 'GBM VaR']])
        ax[0, 0].legend(df[['Normal VaR', 'GBM VaR']].columns, loc="best")
    else:
        ax[0, 0].plot(df[['GBM VaR']])
        ax[0, 0].legend(df[['GBM VaR']].columns, loc="best")
    ax[0, 0].set_title("Parametric VaR")

    ax[0, 1].plot(df[['Historical rel VaR', 'Historical abs VaR']])
    ax[0, 1].set_title('Historical VaR')
    ax[0, 1].legend(df[['Historical rel VaR', 'Historical abs VaR']].columns, loc='best')

    ax[1, 0].plot(df[['MC multi VaR', 'MC one VaR']])
    ax[1, 0].set_title('Monte Carlo VaR')
    ax[1, 0].legend(df[['MC multi VaR', 'MC one VaR']].columns, loc='best')

    ax[1, 1].plot(df)
    ax[1, 1].set_title('All Methods Comparison')
    ax[1, 1].legend(df.columns, loc='best')
    plt.show()

# Plot ES Button, 2 different methods ES and comparison
def Plot_ES(df, period, elevel):
    fig = plt.figure(figsize=(15, 10))
    gs = gridspec.GridSpec(4, 4)
    fig.suptitle('Plot of Portfolio {} day {:.1%} ES'.format(period, elevel), fontsize=20)

    ax1 = fig.add_subplot(gs[0:2, 0:2])
    ax1.plot(df[['Historical rel ES', 'Historical abs ES']])
    ax1.set_title('Historical ES')
    ax1.legend(df[['Historical rel ES', 'Historical abs ES']].columns, loc='best')

    ax2 = fig.add_subplot(gs[0:2, 2:4])
    ax2.plot(df[['MC multi ES', 'MC one ES']])
    ax2.set_title('Monte Carlo ES')
    ax2.legend(df[['MC multi ES', 'MC one ES']].columns, loc='best')

    ax3 = fig.add_subplot(gs[2:4, 1:3])
    ax3.plot(df)
    ax3.set_title('All Methods Comparison')
    ax3.legend(df.columns, loc='best')
    plt.subplots_adjust(hspace=0.5, wspace=0.5)
    plt.show()

# Backtest Button, plot validation results, return average exception numbers of each method
def Plot_Backtest(loss, exceptions, period, level):
    avg_exceptions = exceptions.mean(axis=0)                                    # average number of exceptions in one year

    fig = plt.figure(figsize=(15, 10))
    gs = gridspec.GridSpec(2, 2)
```

```
    fig.suptitle('Validation␣Results␣of␣{}␣day␣{:.1%}␣VaR'.format(period, level), fontsize=20)

    ax1 = fig.add_subplot(gs[0, :])
    ax1.plot(loss)
    ax1.set_title('VaR␣v.s.␣Actual␣Loss')
    ax1.set_ylim(bottom=0)
    ax1.legend(loss.columns, loc='best')

    ax2 = fig.add_subplot(gs[1, :])
    ax2.plot(exceptions)
    ax2.set_title('Number␣of␣Exceptions')
    ax2.legend(exceptions.columns, loc='best')
    ax2.axhline(y=round(252 * (1 - level), 2), ls='--', color='red')
    plt.show()

    return avg_exceptions

# plot number of VaR exceptions
def Plot_exception(bt_result, level):
    expect = round(252 * (1 - level), 2)
    handles = [Line2D([0], [0], color='blue', linestyle='-'), Line2D([0],
               [0], color='red', linestyle='--')]
    labels = ['number␣of␣exceptions', 'expected␣number␣=␣' + str(expect)]

    plt.plot(bt_result)
    plt.title('1␣year␣VaR␣exception,␣p=' + str(level))
    plt.axhline(y=expect, ls='--', color='red')
    plt.legend(handles=handles, labels=labels, loc='best')
    plt.show()


# main read function, output dataframes and adjusted time period
def Read_Main(path, time_start, time_end, l, horizon_period, parameter):
    try:
        option_data = Read_option(path)
        stocks, options = Read_position(path, option_data)

        time_start, time_end = Adjust_option_time(option_data, options, time_start, time_end)
        stock_data, time_start, time_end = Read_stock(stocks, time_start, time_end)

        if bool(options):
            interest_rate = Read_interest(path, time_start, time_end)
        else:
            interest_rate = None

    except (FileNotFoundError, AssertionError, ValueError, NameError,
            RemoteDataError, OverflowError, KeyError) as err:
        raise ValueError(err)

    try:
        p_return, option_data, stock_data, long_p = \
        Portfolio_return(stock_data, option_data, interest_rate,
                         stocks, options, time_start, time_end)
        rho = Calculate_rho(stock_data, l)
        p_hist = Historical_change(p_return, horizon_period)

        if parameter == 'Window':
            gbm_stocks = Calibrate_all(stock_data, l, horizon_period)
            gbm_portfolio = Calibrate(p_return, l, horizon_period)
        elif parameter == 'Exponential':
            gbm_stocks = Calibrate_all_EW(stock_data, l, horizon_period)
```

```
            gbm_portfolio = Calibrate(p_return, l, horizon_period)

    except ValueError as err:
        raise ValueError(err)

    return stocks, options, stock_data, option_data, interest_rate, p_return, long_p, rho, p_hist, \
            time_start, time_end, gbm_stocks, gbm_portfolio

# main calculate function, output four df: VaR, ES, Actual Loss and Exceptions
def Calculate_Main(port_df, port_hist, stock_df, option_df, stocks, options,
                   rho, int_rate, gbm_stocks, gbm_portfolio, V, l, period, vlevel, elevel, long_p):

    gbm_VaR = Parametric_VaR_GBM(gbm_portfolio, V, period, vlevel, long_p)

    his_VaR = Historical_VaR(port_hist, V, l, vlevel, long_p)
    MC_one_VaR = MC_VaR_One(gbm_portfolio, V, period, vlevel, 10000, long_p)
    MC_multi_VaR = MC_VaR_Multi(gbm_stocks, port_df, option_df, stock_df, stocks, options, rho,
                                int_rate, V, l, period, vlevel, 10000, long_p)

    His_ES = Historical_ES(port_hist, V, l, elevel, long_p)
    MC_one_ES = MC_ES_One(gbm_portfolio, V, period, elevel, 10000, long_p)
    MC_multi_ES = MC_ES_Multi(gbm_stocks, port_df, option_df, stock_df, stocks, options, rho,
                              int_rate, V, l, period, elevel, 10000, long_p)
    # combine data
    if bool(stocks):
        normal_VaR = Parametric_VaR_Normal(stock_df, gbm_stocks, stocks, rho, V, l, period, vlevel, long_p)
        VaR_df = pd.concat([MC_multi_VaR, MC_one_VaR, his_VaR, normal_VaR, gbm_VaR], axis=1)
    else:
        VaR_df = pd.concat([MC_multi_VaR, MC_one_VaR, his_VaR, gbm_VaR], axis=1)
    VaR_df = VaR_df.fillna(method='bfill')

    ES_df = pd.concat([MC_multi_ES, MC_one_ES, His_ES], axis=1)
    ES_df = ES_df.fillna(method='bfill')

    exceptions = pd.DataFrame()
    for name in list(VaR_df.columns):
        Backtest = Backtesting(port_df, VaR_df[[name]], V, period, long_p)
        Backtest.columns = [name]
        exceptions = pd.concat([exceptions, Backtest], axis=1)

    v_start = VaR_df.index[0]
    v_end = VaR_df.index[-1]
    p_start = port_df.index.get_loc(v_start)
    p_end = port_df.index.get_loc(v_end)

    price_change = port_df['Price'].values[p_start + period:p_end] - port_df['Price'].values[p_start:p_end - period]
    price_change = np.array(price_change) / np.array(port_df['Price'][p_start:p_end - period]) * V

    if long_p == True:
        loss = pd.DataFrame({'Actual loss':-price_change}, index = VaR_df.index[:-period-1])
        loss = pd.concat([loss, VaR_df.iloc[:-period, :]], axis=1)
    else:
        loss = pd.DataFrame({'Actual loss':price_change}, index = VaR_df.index[:-period-1])
        loss = pd.concat([loss, VaR_df.iloc[:-period, :]], axis=1)

    return VaR_df, ES_df, loss, exceptions

# save calculation results
def Save_Data(VaR, ES, Exceptions, path):
    folder_name = datetime.now().strftime('%Y_%m_%d_%H_%M_%S')
    file_path = os.path.join(path, folder_name)
```

```
    os.mkdir(file_path)
    name1 = os.path.join(file_path, 'VaR.csv')
    VaR = VaR.iloc[::-1]
    VaR.to_csv(name1)
    name2 = os.path.join(file_path, 'ES.csv')
    ES = ES.iloc[::-1]
    ES.to_csv(name2)
    name3 = os.path.join(file_path, 'Exceptions.csv')
    Exceptions = Exceptions.iloc[::-1]
    Exceptions.to_csv(name3)
```

A.4 Codes for GUI

```
# python3
# -*- coding: utf-8 -*-
"""
Created on Wed Dec 11 02:15:50 2019

@author: yz3380@columbia.edu
"""

import os
import pandas as pd
import tkinter as tk
import tkinter.messagebox
import webbrowser
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg, NavigationToolbar2Tk
from tkinter import ttk
from my_utils import *


# GUI setups
AUTHOR_INFO = 'Version 1.3\nYunxiao Zhao, Ophelia Jiang\nyz3380@columbia.edu, wj2289@columbia.edu'
LABEL_NAME_1 = ['Starting Time:', 'Ending Time:', 'Portfolio Value:', 'Est. Method:']
LABEL_NAME_2 = ['VaR Level:', 'ES Level:', 'Window Length:', 'Horizon Period:']
SMALL = ('Arial', 13)
MEDIUM = ('Arial', 16)
WIDTH = 10


def Errormsg(msg):                                                  # global errmsgbox function
    tkinter.messagebox.showwarning("Error", msg)

def Noticemsg(msg):
    tkinter.messagebox.showinfo("Notification", msg)


class RMSystem(tk.Tk):                                              # base class

    def __init__(self, *args, **kwargs):

        tk.Tk.__init__(self, *args, **kwargs)
        self.title("GR5320 Project v1.3")
        #tk.Tk.wm_title(self, 'GR5320 Project v1.0')
        self.geometry("640x480")
```

```
        container = tk.Frame(self)
        container.pack()

        self.frames = {}

        for F in (IndexPage, CalPage):
            frame = F(container, self)
            self.frames[F] = frame
            frame.grid(row=0, column=0, sticky='nsew')

        self.show_frame(IndexPage)

    def show_frame(self, cont):
        frame = self.frames[cont]
        frame.tkraise()

class IndexPage(tk.Frame):                                              # index page

    def __init__(self, parent, controller):

        tk.Frame.__init__(self, parent)

        s = ttk.Style()                                                 # set ttk button styles
        s.configure('TButton', font=('Arial', 18))
        s.configure('my.TButton', font=('Arial', 16))
        s.configure('Red.TButton', font=('Arial', 18), foreground="red")

        Name = tk.Label(self, text="Risk␣Calculation␣System", font=("Arial", 32, 'bold'), fg="blue")
        Name.grid(row=0, column=0, rowspan=3, columnspan=4, sticky="NWES", ipadx=12, pady=20)

        Menu = tk.Label(self, text="Select␣a␣function:", font=("Arial", 12))
        Menu.grid(row=3, column=1, columnspan=2, sticky="S", pady=10)

        GoWin = ttk.Button(self, text="Start", style = 'TButton', width=15,
                           command=lambda: controller.show_frame(CalPage))
        GoWin.grid(row=4, column=1, columnspan=2, ipadx=20, ipady=10, pady=5)

        GoMac = ttk.Button(self, text="Help", style = 'TButton', width=15,
                             command=lambda: self.Open_Help())
        GoMac.grid(row=5, column=1, columnspan=2, ipadx=20, ipady=10, pady=5)

        Quit = ttk.Button(self, text="Quit", style='Red.TButton', width=15, command=lambda: quit())
        Quit.grid(row=6, column=1, columnspan=2, ipadx=20, ipady=10, pady=5)

        Author = tk.Label(self, text=AUTHOR_INFO, font=("Arial", 8), fg="grey")
        Author.grid(row=7, column=1, columnspan=2, sticky="NWES", pady=20)

        self.grid_rowconfigure(3, minsize=75)

    def Open_Help(self):
        url = "https://github.com/yz3380/5320Project"
        webbrowser.open(url, new=1)

class CalPage(tk.Frame):                                                # calculation page

    def __init__(self, parent, controller):

        tk.Frame.__init__(self, parent)
                                                                # initialize variables
        self.path = os.getcwd()
```

```
self.ready_to_plot = False

self.Time_start = pd.Timestamp('1890-8-9')
self.Time_end = pd.Timestamp('2077-7-7')
self.VaR_p = 0.99
self.ES_p = 0.975
self.Initial = 10000.0
self.Length = 5.0
self.Horizon = 5
self.Method = 'Window'

Widgets = {}                                                    # initialize input widgets

Instruc = tk.Label(self, text="Please␣enter␣valid␣inputs:", font=SMALL)
Instruc.grid(row=0, column=0, columnspan=2, pady=10, sticky="WS")

for i, parameter in enumerate(LABEL_NAME_1):
    label = ttk.Label(self, text=parameter, font=SMALL, width=WIDTH+3,
                      foreground='blue', anchor="e")
    label.grid(row=i + 1, column=0, pady=10)

for i, parameter in enumerate(LABEL_NAME_2):
    label = ttk.Label(self, text=parameter, font=SMALL, width=WIDTH+3,
                      foreground='blue', anchor="e")
    label.grid(row=i + 1, column=2, pady=10)

entry = tk.Entry(self, font=MEDIUM, textvar=tk.StringVar(), width=WIDTH, bd=5)
entry.insert(0, '{:%Y-%m-%d}'.format(self.Time_start))
Widgets["start"] = entry
entry.grid(row=1, column=1, padx=5, pady=10)

entry = tk.Entry(self, font=MEDIUM, textvar=tk.StringVar(), width=WIDTH, bd=5)
entry.insert(0, '{:%Y-%m-%d}'.format(self.Time_end))
Widgets["end"] = entry
entry.grid(row=2, column=1, padx=5, pady=10)

entry = tk.Entry(self, font=MEDIUM, textvar=tk.DoubleVar(), width=WIDTH, bd=5)
entry.delete(0, tk.END)
entry.insert(0, self.Initial)
Widgets["initial"] = entry
entry.grid(row=3, column=1, padx=5, pady=10)

entry = tk.Listbox(self, selectmode="SINGLE", font=SMALL, width=WIDTH+4, bd=3, height=2)
entry.insert(1, "Window")
entry.insert(2, "Exponential")
entry.select_set(0)
entry.configure(exportselection=False)
Widgets["method"] = entry
entry.grid(row=4, column=1, padx=5, pady=5)

entry = tk.Entry(self, font=MEDIUM, textvar=tk.DoubleVar(), width=WIDTH, bd=5)
entry.delete(0, tk.END)
entry.insert(0, self.VaR_p)
Widgets["vlevel"] = entry
entry.grid(row=1, column=3, padx=5, pady=10)

entry = tk.Entry(self, font=MEDIUM, textvar=tk.DoubleVar(), width=WIDTH, bd=5)
entry.delete(0, tk.END)
entry.insert(0, self.ES_p)
Widgets["elevel"] = entry
entry.grid(row=2, column=3, padx=5, pady=10)
```

```
        entry = tk.Entry(self, font=MEDIUM, textvar=tk.DoubleVar(), width=WIDTH, bd=5)
        entry.delete(0, tk.END)
        entry.insert(0, self.Length)
        Widgets["length"] = entry
        entry.grid(row=3, column=3, padx=5, pady=10)

        entry = tk.Entry(self, font=MEDIUM, textvar=tk.IntVar(), width=WIDTH, bd=5)
        entry.delete(0, tk.END)
        entry.insert(0, self.Horizon)
        Widgets["period"] = entry
        entry.grid(row=4, column=3, padx=5, pady=5)
                                                                                # Intialize function buttons
        Clear_Btn = ttk.Button(self, text="Default", style='TButton', width=8,
                               command=lambda: self.Clear(Widgets))
        Clear_Btn.grid(row=5, column=0, ipadx=5, ipady=5, pady=10, sticky = 's')

        Cal_Btn = tk.Button(self, text="Calculate", font=('Arial', 20, 'bold'), width=8,
                            fg="white", bg="light␣green", command=lambda: self.Main(Widgets))
        Cal_Btn.grid(row=5, column=3, pady=10)

        Port_Btn = ttk.Button(self, text="Portfolio", style='TButton', width=8,
                            command=lambda: self.Plot_Position())
        Port_Btn.grid(row=6, column=0, ipadx=5, ipady=5)

        VaR_Btn = ttk.Button(self, text="Plot␣VaR", style='TButton', width=8,
                            command=lambda: self.Plot_VaR())
        VaR_Btn.grid(row=6, column=1, ipadx=5, ipady=5)

        ES_Btn = ttk.Button(self, text="Plot␣ES", style='TButton', width=8,
                            command=lambda: self.Plot_ES())
        ES_Btn.grid(row=6, column=2, ipadx=5, ipady=5)

        Bt_Btn = ttk.Button(self, text="Backtest", style='TButton', width=8,
                            command=lambda: self.Plot_Backtest())
        Bt_Btn.grid(row=6, column=3, ipadx=5, ipady=5)

        Back_Btn = ttk.Button(self, text="Back", style='TButton', width=8,
                            command=lambda: controller.show_frame(IndexPage))
        Back_Btn.grid(row=7, column=0, ipadx=5, ipady=5, pady=10)

        Save_Btn = ttk.Button(self, text="Save", style='TButton', width=8,
                            command=lambda: self.Save_Data())
        Save_Btn.grid(row=7, column=2, ipadx=5, ipady=5, pady=10)

        Quit_Btn = ttk.Button(self, text="Quit", style='Red.TButton', width=8,
                            command=lambda: quit())
        Quit_Btn.grid(row=7, column=3, ipadx=5, ipady=5, pady=10)

    def Clear(self, widgets):
        widgets['start'].delete(0, tk.END)
        widgets['start'].insert(0, '1890-8-9')
        widgets['end'].delete(0, tk.END)
        widgets['end'].insert(0, '2077-7-7')
        widgets['vlevel'].delete(0, tk.END)
        widgets['vlevel'].insert(0, 0.99)
        widgets['elevel'].delete(0, tk.END)
        widgets['elevel'].insert(0, 0.975)
        widgets['initial'].delete(0, tk.END)
        widgets['initial'].insert(0, 10000.0)
        widgets['length'].delete(0, tk.END)
```

```
        widgets['length'].insert(0, 5.0)
        widgets['period'].delete(0, tk.END)
        widgets['period'].insert(0, 5)
        widgets['method'].selection_clear(0, 2)
        widgets['method'].select_set(0)

    def Main(self, widgets):                                                    # main function

        try:
            assert(widgets['method'].curselection()!=())
        except AssertionError:
            Errormsg('Please␣select␣an␣estimation␣method!')
            return

        try:
            self.Time_start = pd.Timestamp(widgets['start'].get())
            self.Time_end = pd.Timestamp(widgets['end'].get())
            self.VaR_p = float(widgets['vlevel'].get())
            self.ES_p = float(widgets['elevel'].get())
            self.Initial = float(widgets['initial'].get())
            self.Length = float(widgets['length'].get())
            self.Horizon = int(widgets['period'].get())
            self.Method = widgets['method'].get(widgets['method'].curselection())
                                                                                # check input validity
            Check_input(self.Time_start, self.Time_end, self.VaR_p,
                        self.ES_p, self.Initial, self.Length, self.Horizon, self.Method)

        except (ValueError, AssertionError):
            Errormsg('Invalid␣inputs!␣Please␣check␣your␣setups')
            return

        try:
            self.Stocks, self.Options, self.Stock_Data, self.Option_Data, \
            self.Interest_Rate, self.P_Return, self.Long_P, self.Rho, self.P_Hist, \
            self.Time_start, self.Time_end, self.GBM_Stocks, self.GBM_Portfolio = \
            Read_Main(self.path, self.Time_start, self.Time_end,
                      self.Length, self.Horizon, self.Method)
        except ValueError as err:
            Errormsg(err)
            return

        self.VaR, self.ES, self.Loss, self.Exceptions = \
        Calculate_Main(self.P_Return, self.P_Hist, self.Stock_Data,
                       self.Option_Data, self.Stocks, self.Options,
                       self.Rho, self.Interest_Rate, self.GBM_Stocks,
                       self.GBM_Portfolio, self.Initial, self.Length,
                       self.Horizon, self.VaR_p, self.ES_p, self.Long_P)

        self.ready_to_plot = True
        widgets['start'].delete(0, tk.END)
        widgets['start'].insert(0, '{:%Y-%m-%d}'.format(self.Time_start))
        widgets['end'].delete(0, tk.END)
        widgets['end'].insert(0, '{:%Y-%m-%d}'.format(self.Time_end))

        Noticemsg('Calculation␣complete!\nTime␣period␣automatically␣adjusted␣to\n{}␣~␣{}.'
                  .format(self.Time_start.strftime('%Y-%m-%d'), self.Time_end.strftime('%Y-%m-%d')))

        if (self.P_Return['Price'].min()==0.01 or self.P_Return['Price'].max()==-0.01):
            Errormsg('WARNING:␣␣Portfolio␣value␣sign␣changes␣during␣the␣period,␣\nresults␣may␣not␣be␣reliable!!!')

    def Plot_Position(self):
```

```
    if not self.ready_to_plot:
        Errormsg('Please␣calculate␣result␣first')
        return
    port_df, stock_df, option_df, stocks, options, int_rate, gbm_df = \
    self.P_Return, self.Stock_Data, self.Option_Data, self.Stocks, \
    self.Options, self.Interest_Rate, self.GBM_Portfolio

    df = pd.DataFrame()
    if isinstance(stock_df, pd.core.frame.DataFrame):
        df['Portfolio'] = stock_df['Price']
    else:
        for ticker in stocks.keys():
            df[ticker] = stock_df[ticker]['Price']

    name = []
    value = []

    for ticker in stocks:
        name.append(ticker + ':␣' + str(stocks[ticker]))
        value.append(abs(stock_df[ticker].iloc[0, 0] * stocks[ticker]))
    for ticker in options:
        name.append(ticker + ':␣'+ str(options[ticker][0]))
        iv_index = 3 if options[ticker][1] == 'call' else 0                  # implied volatility index
        iv_lib = dict(zip([3, 6, 12], [1, 2, 3]))
        iv_index = iv_index + iv_lib[options[ticker][2]]
        value.append(abs(float(Black_Scholes(option_df[ticker].iloc[0, 0],
                                             option_df[ticker].iloc[0, 0],
                                             int_rate.values[0],
                                             option_df[ticker].iloc[0,iv_index],
                                             options[ticker][2]/12, options[ticker][1])
                               * options[ticker][0])))

    explode = [0.1] * len(name)

    root2 = tk.Toplevel()
    root2.title('Portfolio␣Info')

    fig, ax = plt.subplots(2, 2, figsize=(12, 8))
    canvas2 = FigureCanvasTkAgg(fig, master=root2)
    canvas2.get_tk_widget().pack(side=tk.TOP, fill=tk.BOTH, expand=1.0)

    fig.suptitle('Visualization␣of␣Portfolio␣Info', fontsize=20)
    ax[0, 0].pie(value, explode=explode, labels=name, shadow=True)
    ax[0, 0].axis('equal')
    ax[0, 0].legend(name, title="Assets", loc="best")
    ax[0, 0].set_title("Initial␣Asset␣Allocation")
    if bool(stocks):
        ax[0, 1].plot(df)
        ax[0, 1].set_title('Historical␣Stock␣Prices')
        ax[0, 1].legend(df.columns, loc='best')
    ax[1, 0].plot(port_df['Price'])
    ax[1, 0].set_title('Unit␣Portfolio␣Value')
    ax[1, 1].plot(gbm_df)
    ax[1, 1].set_title('Portfolio␣Return␣Drift␣and␣Volatility')
    ax[1, 1].legend(gbm_df.columns, loc='best')

    toobar = NavigationToolbar2Tk(canvas2, root2)
    toobar.update()
    canvas2.draw()

def Plot_VaR(self):
```

```
        if not self.ready_to_plot:
            Errormsg('Please␣calculate␣result␣first')
            return
        df, stocks, period, vlevel = self.VaR, self.Stocks, self.Horizon, self.VaR_p

        root2 = tk.Toplevel()
        root2.title('Analysis␣of␣VaR')
        fig, ax = plt.subplots(2, 2, figsize=(12, 8))
        canvas2 = FigureCanvasTkAgg(fig, master=root2)
        canvas2.get_tk_widget().pack(side=tk.TOP, fill=tk.BOTH, expand=1.0)
        fig.suptitle('Plot␣of␣Portfolio␣{}␣day␣{:.1%}␣VaR'.format(period, vlevel), fontsize=20)
        if bool(stocks):
            ax[0, 0].plot(df[['Normal␣VaR', 'GBM␣VaR']])
            ax[0, 0].legend(df[['Normal␣VaR', 'GBM␣VaR']].columns, loc="best")
        else:
            ax[0, 0].plot(df[['GBM␣VaR']])
            ax[0, 0].legend(df[['GBM␣VaR']].columns, loc="best")
        ax[0, 0].set_title("Parametric␣VaR")

        ax[0, 1].plot(df[['Historical␣rel␣VaR', 'Historical␣abs␣VaR']])
        ax[0, 1].set_title('Historical␣VaR')
        ax[0, 1].legend(df[['Historical␣rel␣VaR', 'Historical␣abs␣VaR']].columns, loc='best')

        ax[1, 0].plot(df[['MC␣multi␣VaR', 'MC␣one␣VaR']])
        ax[1, 0].set_title('Monte␣Carlo␣VaR')
        ax[1, 0].legend(df[['MC␣multi␣VaR', 'MC␣one␣VaR']].columns, loc='best')

        ax[1, 1].plot(df)
        ax[1, 1].set_title('All␣Methods␣Comparison')
        ax[1, 1].legend(df.columns, loc='best')

        toobar = NavigationToolbar2Tk(canvas2, root2)
        toobar.update()
        canvas2.draw()

    def Plot_ES(self):
        if not self.ready_to_plot:
            Errormsg('Please␣calculate␣result␣first')
            return
        df, period, elevel = self.ES, self.Horizon, self.ES_p

        root2 = tk.Toplevel()
        root2.title('Analysis␣of␣ES')
        fig = plt.figure(figsize=(12, 8))

        canvas2 = FigureCanvasTkAgg(fig, master=root2)
        canvas2.get_tk_widget().pack(side=tk.TOP, fill=tk.BOTH, expand=1.0)

        gs = gridspec.GridSpec(4, 4)
        fig.suptitle('Plot␣of␣Portfolio␣{}␣day␣{:.1%}␣ES'.format(period, elevel), fontsize=20)

        ax1 = fig.add_subplot(gs[0:2, 0:2])
        ax1.plot(df[['Historical␣rel␣ES', 'Historical␣abs␣ES']])
        ax1.set_title('Historical␣ES')
        ax1.legend(df[['Historical␣rel␣ES', 'Historical␣abs␣ES']].columns, loc='best')

        ax2 = fig.add_subplot(gs[0:2, 2:4])
        ax2.plot(df[['MC␣multi␣ES', 'MC␣one␣ES']])
        ax2.set_title('Monte␣Carlo␣ES')
        ax2.legend(df[['MC␣multi␣ES', 'MC␣one␣ES']].columns, loc='best')
```

```
        ax3 = fig.add_subplot(gs[2:4, 1:3])
        ax3.plot(df)
        ax3.set_title('All Methods Comparison')
        ax3.legend(df.columns, loc='best')
        plt.subplots_adjust(hspace=0.5, wspace=0.5)

        toobar = NavigationToolbar2Tk(canvas2, root2)
        toobar.update()
        canvas2.draw()

    def Plot_Backtest(self):
        if not self.ready_to_plot:
            Errormsg('Please calculate result first')
            return
        loss, exceptions, period, level = self.Loss, self.Exceptions, self.Horizon, self.VaR_p

        avg_exceptions = exceptions.mean(axis=0)                                    # average number of exceptions in one year
        Noticemsg("The {} method has the lowest average number of exceptions: {:.2f} days in one year."
                  .format(avg_exceptions.idxmin(), min(avg_exceptions)))

        root2 = tk.Toplevel()
        root2.title('Validation Result')
        fig = plt.figure(figsize=(12, 8))
        canvas2 = FigureCanvasTkAgg(fig, master=root2)
        canvas2.get_tk_widget().pack(side=tk.TOP, fill=tk.BOTH, expand=1.0)

        gs = gridspec.GridSpec(2, 2)
        fig.suptitle('Validation Results of {} day {:.1%} VaR'.format(period, level), fontsize=20)

        ax1 = fig.add_subplot(gs[0, :])
        ax1.plot(loss)
        ax1.set_title('VaR v.s. Actual Loss')
        ax1.set_ylim(bottom=0)
        ax1.legend(loss.columns, loc='best')

        ax2 = fig.add_subplot(gs[1, :])
        ax2.plot(exceptions)
        ax2.set_title('Number of Exceptions')
        ax2.legend(exceptions.columns, loc='best')
        ax2.axhline(y=round(252 * (1 - level), 2), ls='--', color='red')

        toobar = NavigationToolbar2Tk(canvas2, root2)
        toobar.update()
        canvas2.draw()

    def Save_Data(self):
        if not self.ready_to_plot:
            Errormsg('Please calculate result first')
            return
        Save_Data(self.VaR, self.ES, self.Exceptions, self.path)
        Noticemsg("Results successfully saved")


def quit():
    app.destroy()
    plt.close('all')


app = RMSystem()                                                                    # run app in the center of the screen.
positionRight = int(app.winfo_screenwidth()/2 - 320)
positionDown = int(app.winfo_screenheight()/2 - 240)
```

```
app.geometry("+{}+{}".format(positionRight, positionDown))
app.mainloop()
```