Software Design Documentation

Jordan Boucher - jpb2232

December 2023

Contents

1 Introduction

This document focuses on the software needed to execute our equity portfolio VaR model. We will begin with a high-level overview of the program in the Software Architecture section. Although this section refrains from delving into specifics, it serves as a effective first pass of the underlying code. The subsequent Module Documentation section will offer detailed insights into the specific functions and modules that constitute our system. Specifically, we will explore two key modules: Calculating VaR and Backtesting, providing details on the essential functions contained within each. Additionally, we will shed light on the User Interface, describing the key functions required for end-user comprehension. The final section, Data Flow Documentation, will delve into a more intricate explanation of the interactions between functions during the execution of the code, clarifying the dynamic flow within the system.

2 Software Architecture

All of the code is contained in a singular Jupyter Notebook, which is running Python as the underlying language. The document contains hidden code, which is not necessary for the end user to see or understand, but may be of interest to researches. The hidden cells contain the following:

- **Packages**: This cell contains all of the packages necessary for the code to run. Note that the code assumes that the user has pip installed (if necessary) all of these packages. In particular, if the user has never used the *yfinance* package, which is a user friendly way of pulling data off of Yahoo Finance's API, he or she can run the command *!pip install yfinance*
- **VaR Module**: This cell is a collection of many functions which, together, calculate VaR and ES using the various methods described in the Model Documentation. The basic flow of the functions if (1) tranforming the data (see 3.3 User Interface), (2) fitting the data to GBM parameters, and (3) calculating VaR and ES.
- **Backtesting Module**: This cell is also a collection of several functions, these being used to backtest the VaR results.

The cells below are visible to the user by default. Their first task will be to create usable data for the system. This will require them to enter in several choices including but not limited to: the stocks under consideration, their positions, and the time frame of interest. The subsequent cell is where the user will specify which types of VaR and ES they want, and they will receive graphs of historical values. Finally, they will be able to perform backtesting on their portfolio and see how the VaR metric would have performed had they used it in the past.

3 Module Documentation

3.1 Calculating VaR and ES

The purpose of this module is to transform the data, fit the data to Geometric Brownian Motion, and then calculate VaR. To start, we include a function of retrieving the necessary data from Yahoo Finance. NOTE: it is HIGHLY recommended that the user use this method rather can providing their own .csv or .xlsx file of data.[1]

```
get_percent_returns(stocks,positions,units,start_date,
    end_date,data=False):

    - stocks: list of strings which represent tickers
    - positions: list of strings which are either "Long",
        "Short", "Put", or "Call" corresponding to the
        position of the portfolio in that stock
    - units: list of integers and lists; if position for a
       given stock is "Long" or "Short", the
       corresponding element should be the number of
       shares. If the position is "Put" or "Call",
       then the element should be a list with the
       following form (all integers): [contracts,
       volatility, time to maturity, interest rate]
    - start_date: string of the format "YYYY-MM-DD",
       corresponds to the first data considered for
       VaR analysis
    - end_date: string of the format "YYYY-MM-DD",
       corresponds to the first data considered for
       VaR analysis
    - data: list of pandas dataframes which is a manual
       alternative to using Yahoo Finance to pull data
       . NOT RECCOMENDED. Each dataframe must match
       the form of the dataframe created by running pd
       .read_csv("HD-bloomberg.csv")

    return portfolio_percent_returns,
       stock_percent_returns, stock_proportions,
       position_values

    - portfolio_percent_returns: pandas series with
       index equal to the date (timestamp) and values (
       float) equal to the daily percent return to the
       portfolio
```

[1]The function will work for HD and UNH data, so there is no need to use the files from Courseworks.

- stock_percent_returns: *list of pandas series* of length equal to the number of stocks where each pandas series has index equal to the date (*timestamp*) and values (*float*) equal to the daily percent return of the corresponding stock
- stock_proportions: *list of pandas series* of length equal to the number of stocks with index equal to the date (*timestamp*) and values (*float*) equal to the percent of portfolio holdings in that corresponding stock/option
- position_values: *list of pandas series* of length equal to the number of stocks with index equal to the date (*timestamp*) and values (*float*) equal to the daily dollar holdings in the corresponding stock/option

Now that the user has the data in the proper format, they can now calculate the VaR of their choosing. There are separate functions for parametric VaR, Monte Carlo VaR, and historical VaR.

Parametric VaR We start with parametric VaR, which contains the option to do windowing or exponential weighting for the geometric Brownian motion fitting. Note that the "=" for some inputs represents a default value.

```
get_VaR(percent_returns, cash, years=False,p=.99,back
    =252*10,lamb=False):
```

- percent_returns: *pandas series* which is the portfolio_percent_returns output from the get_percent_returns function
- cash: *integer* which corresponds to the amount of money being invested in the portfolio each day. This is so that the VaRs are comparable over time
- years: *integer* which refers to the number of years of data being used for the GBM parameter fitting via windowing. If doing exponential weighting, set equal to "False"
- p: *float* corresponding to the confidence level of the VaR
- back: *integer* corresponding to the number of days at the beginning of the data set for which VaR will not be calculated. Must be greater than 252*years if doing windowing
- lamb: *float* which corresponds to λ if doing exponential weighting. If doing windowing, set equal to "False"

```
    return long_VaR

    - long_VaR: list of length equal to the length of
       the percent_returns minus back corresponding to
        daily VaR
```

Note that this function only has the capability of calculating long VaR. For finding the parametric VaR of a short position in the portfolio or long ES or short ES, use the following functions which are defined analogously:

```
get_short_VaR(percent_returns, cash, years=False,p
   =.99,back=252*10,lamb=False):
    return short_VaR

get_ES(percent_returns, cash, years=False,p=.99,back
   =252*10,lamb=False):
    return long_ES

get_short_ES(percent_returns, cash, years=False,p=.99,
   back=252*10,lamb=False):
    return short_ES
```

Monte Carlo VaR Now, we move on to Monte Carlo VaR. One function contains all possible combinations: long portfolio VaR, long portfolio ES, short portfolio VaR, and short portfolio ES:

```
get_MC_VaR_ES(percent_returns_lst,
   stock_proportions_lst,position_values_lst, cash,
   units, positions, sims, years=False,p=.99,back
   =252*10,lamb=False):

    - percent_returns_lst: list of pandas series which is
       the stock_percent_returns output from the
       get_percent_returns function
    - stock_proportions_lst: list of pandas series which
       is the stock_proportions output from the
       get_percent_returns function
    - position_values_lst: list of pandas series which is
       the position_values output from the
       get_percent_returns function
    - cash: integer which corresponds to the amount of
       money being invested in the portfolio each day.
        This is so that the VaRs are comparable over
       time
    - units: list of integers and lists which is same as
       what was entered for units into the get_percent
        returns function
```

```
- positions: list of strings which is same as what
    was entered for positions into the get_percent
    returns function
-sims: integer corresponding to the number of
    simulations completed for each stock on each
    day of the dataset
- years: integer which refers to the number of
    years of data being used for the GBM parameter
    fitting via windowing. If doing exponential
    weighting, set equal to "False"
- p: float corresponding to the confidence level of
     the VaR
- back: integer corresponding to the number of days
     at the beginning of the data set for which VaR
     will not be calculated. Must be greater than
    252*years if doing windowing
- lamb: float which corresponds to λ if doing
    exponential weighting. If doing windowing, set
    equal to "False"

return long_VaR, long_ES, short_VaR, short_ES

- long_VaR: list of length equal to the length of
    the percent_returns minus back corresponding to
     daily VaR if going long the portfolio
- long_ES: list of length equal to the length of
    the percent_returns minus back corresponding to
     daily ES if going long the portfolio
- short_VaR: list of length equal to the length of
    the percent_returns minus back corresponding to
     daily VaR if going short the portfolio
- short_ES: list of length equal to the length of
    the percent_returns minus back corresponding to
     daily ES if going short the portfolio
```

Note that this function takes a very long time to run. For a portfolio containing two positions, one in a stock and one in an option, and 10,000 simulations, it takes 4.6 hours to complete. It is recommended, then, to do 1,000 simulations or less.

Historical VaR We conclude with historical VaR, which is the simplest function of the three. Like Monte Carlo VaR, one function contains all combinations of long and short and VaR and ES:

```
get_historical_VaR(position_values_lst, cash,
    positions, years, p=.99, back=252*10):
```

```
- position_values_lst: list of pandas series which is
    the position_values output from the
    get_percent_returns function
- cash: integer which corresponds to the amount of
    money being invested in the portfolio each day.
     This is so that the VaRs are comparable over
    time
- positions: list of strings which is same as what
    was entered for positions into the get_percent
    returns function
- years: integer which refers to the number of
    years of data being used for the GBM parameter
    fitting via windowing. If doing exponential
    weighting, set equal to "False"
- p: float corresponding to the confidence level of
     the VaR
- back: integer corresponding to the number of days
     at the beginning of the data set for which VaR
     will not be calculated. Must be greater than
    252*years if doing windowing

return long_VaR, long_ES, short_VaR, short_ES

- long_VaR: list of length equal to the length of
    the percent_returns minus back corresponding to
     daily VaR if going long the portfolio
- long_ES: list of length equal to the length of
    the percent_returns minus back corresponding to
     daily ES if going long the portfolio
- short_VaR: list of length equal to the length of
    the percent_returns minus back corresponding to
     daily VaR if going short the portfolio
- short_ES: list of length equal to the length of
    the percent_returns minus back corresponding to
     daily ES if going short the portfolio
```

3.2 Backtesting

Now that the user has a historical time series of VaR values, they can now backtest it and see how often, historically, were there losses that exceeded those predicted by VaR (any type). The following function calculating a moving window of number of VaR days over the past year (252 trading days):

```
get_year_window(long_VaR_ts,short_VaR_ts,
    position_values_lst,positions,cash):
```

```
- long_VaR_ts: list containing 5-day p% VaR values
    over time; long_VaR output from any of the
    above VaR functions
- short_VaR_ts: list containing 5-day p% short VaR
    values over time; short_VaR output from any of
    the above VaR functions
- position_values_lst: list of pandas series which is
    the position_values output from the
    get_percent_returns function
- positions: list of strings which is same as what
    was entered for positions into the get_percent
    returns function
- cash: integer which corresponds to the amount of
    money being invested in the portfolio each day.
     This is so that the VaRs are comparable over
    time

return long_VaR, long_ES, short_VaR, short_ES

- long_VaR: list of length equal to the length of
    the percent_returns minus back corresponding to
     daily VaR if going long the portfolio
- long_ES: list of length equal to the length of
    the percent_returns minus back corresponding to
     daily ES if going long the portfolio
- short_VaR: list of length equal to the length of
    the percent_returns minus back corresponding to
     daily VaR if going short the portfolio
- short_ES: list of length equal to the length of
    the percent_returns minus back corresponding to
     daily ES if going short the portfolio
```

4 Data Flow Documentation

In this section, we give an example of the flow of data through the system. The User Input includes all of the following:

1. A list of stock tickers
2. A list of positions (long, short, call, or put)
3. The number of shares held in each of the stocks
4. The volatility, time to maturity, and interest rate of the at-the-money options
5. The start and end dates of the anlysis

6. Either the number of years used for windowing or the λ used for exponential weighting

Also, the user can optionally provide:

1. Historical price data of their stocks
2. The constant cash holding of the portfolio over time
3. A specific p-value for VaR for ES (otherwise assumed to be 99% and 97.5%, respectively)
4. The back period over which VaR is not calculated
5. The number of Monte Carlo simulations conducted

If the user completes the entire parametric VaR analysis using the functions described above the data will move as follows: