Imperial College London

COMPUTATIONAL FINANCE WITH C++

IMPERIAL COLLEGE BUSINESS SCHOOL

DEPARTMENT OF FINANCE

Markowitz Model & Rolling Window Back-Testing

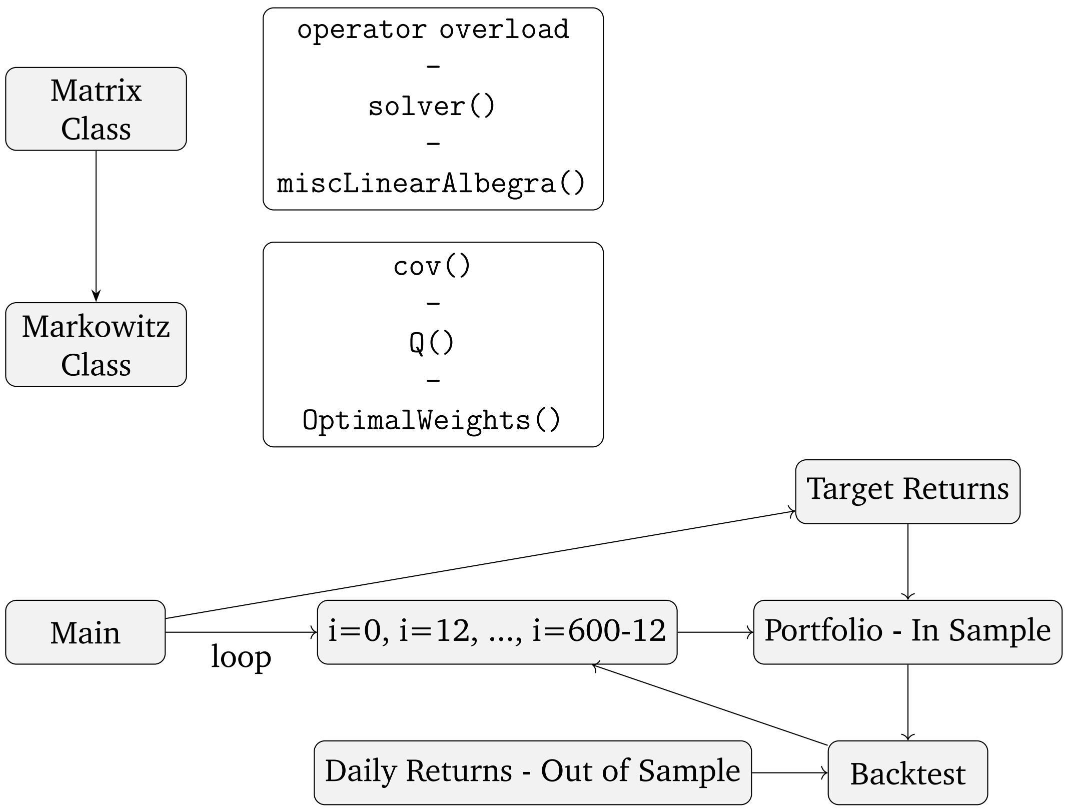
Author:
Edward Peterson (CID: 01502703)

Date: May 29, 2024

1 Software Structure

www.github.com/ep4518/CFcrsw

- no use of polymorphism
- read_data.h and read_data.cpp unchanged
- defined type Vector and Lattice as vector<double> and vector<vector<double>>
- defined class "Matrix" holds a lattice and implements rudimentary linear algebra with multiple constructors available e.g. (rows, columns), (Lattice), ().
  - operator overload for multiplication, addition, subtraction, unary negative and also for scalar equivalent operations
  - operator overload for splcing, along with functionallity for insertion, printing, retrieval, shape etc.
  - ultimately building towards implementing the Conjugate Gradient Descent Solver.
- implemented numpy-like horizontal and vertical stacking of Matrix triples
- Markowitz class for defining a portfolio with optimal asset weightings
  - mean() - average returns for each asset over sample period - $\bar{r}_i = \frac{1}{n}\Sigma_{k=1}^{n} r_{i,k}$
  - cov() - covariance of asset returns - $\Sigma_{ij} = \frac{1}{n-1}\sum_{k=1}^{n}(r_{i,k}-\bar{r}_i)(r_{j,k}-\bar{r}_j)$
  - b(double target_return), Q() - vstack(hstack, hstack, hstack)
  - optimal_weights(): $Qx = b$
- backtesting function implemented for each iteration in the rolling window
- wrote array of result structs in csv form for plotting with python - write_data.h



2 Evaluation

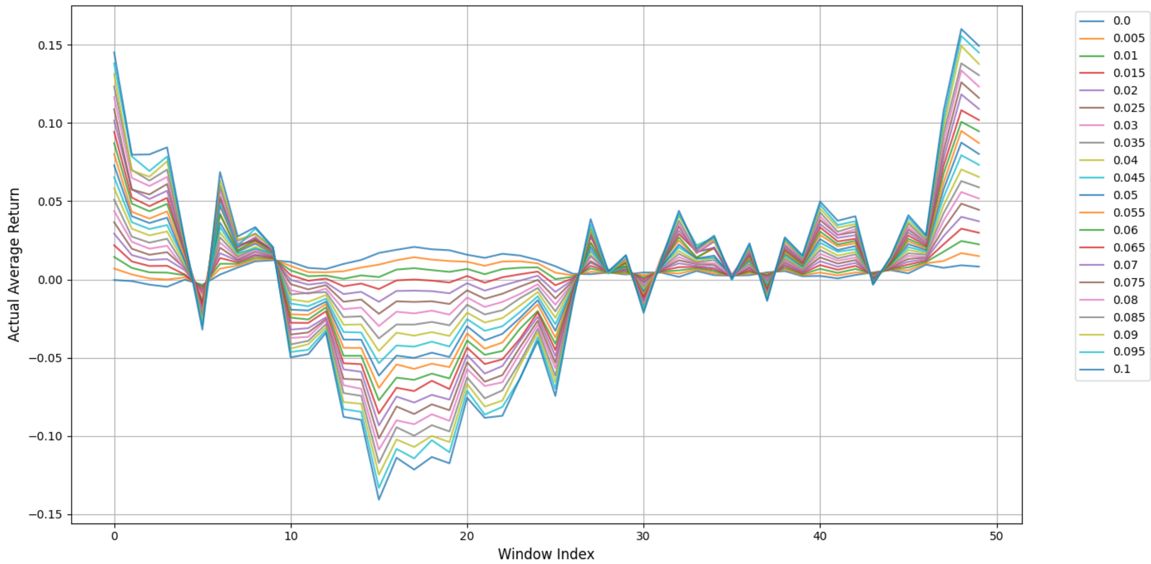


Figure 1: Realised Rolling Window Average OOS Return for each Target

Fig. 1 portrays the rolling window realised average return for each target over the course of the backtest. There are 50 periods in the backtest indexed on the x-axis of the plot. Out of sample performance of the Markowtiz model is poor. The cumulative performance over the enitre domain is worse the larger the target return becomes.

To evaluate the accuracy of our Markowitz implementation, we contrast with a numpy analog, where the use of the linalg library provides a relaible solver to compare the accuracy of the conjugate gradient descent algorithm (CGD) in cpp. Fig. 2 shows the drastic impact of the fractional differences in optimal weights we see whilst solving with CGD, on the overall performance during the backtesting phase. Following extensive debugging, these differences were also observed in the Matlab conjgrad.m implementation. Any CGD solver used had $||Qx-b|| \approx e^{-5}$ for the range of target returns, where as for prebuilt solvers the norm value usually lies in the range $\approx e^{-11}$.

In both cases, the Markowitz portfolio effectively captures the market Beta, and produces returns that scale consistently with the amount of risk taken by the investor (where the magnitude and direction of those returns is dictated by those of the market).

Figure 3 demonstrates the Sharpe ratio of the portfolios for an example backtest period. In general, we observed this classic efficient frontier shape in most periods, allowing for inversion in periods of negative market returns.

Figure 4 portrays a cumulative portfolio return next to its constituent assets. As

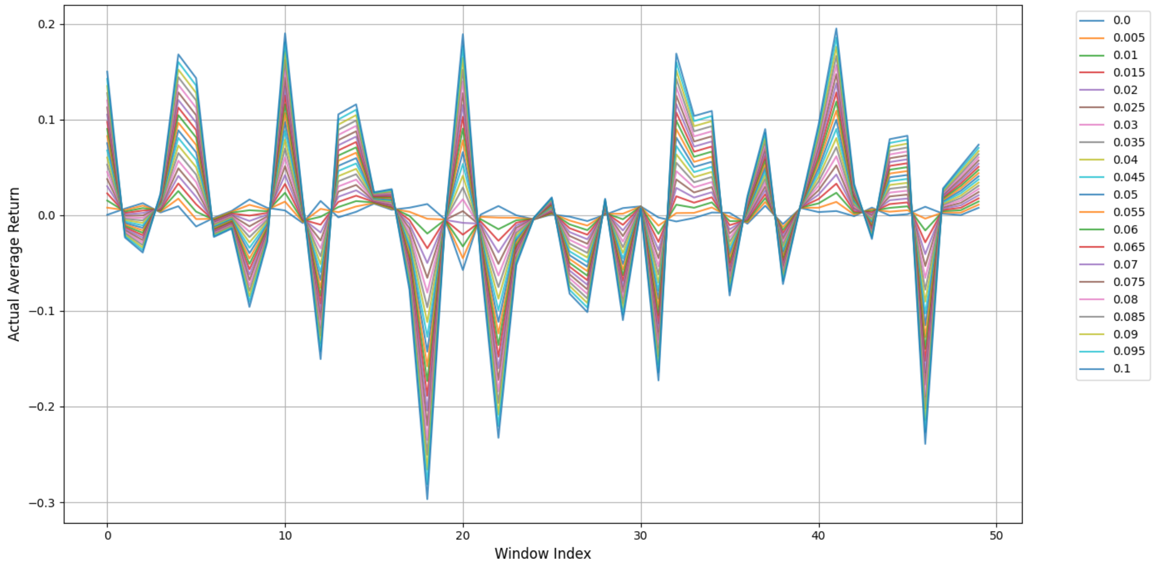


Figure 2: Realised Rolling Window Average OOS Return for each Target - Python

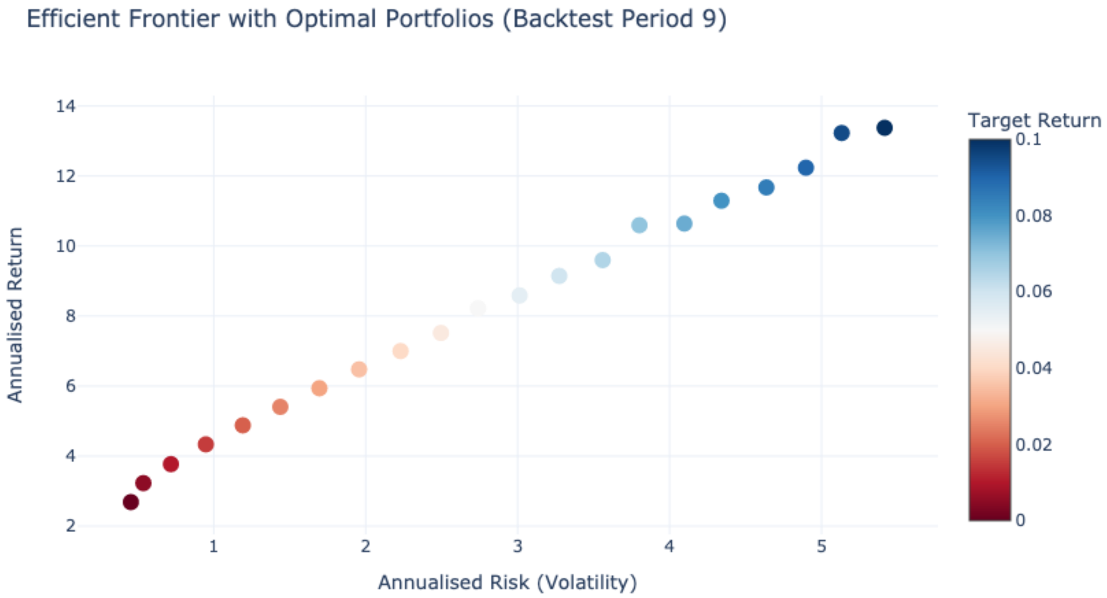


Figure 3: Efficient Frontier with Optimal Portfolios (Backtest Period 9)

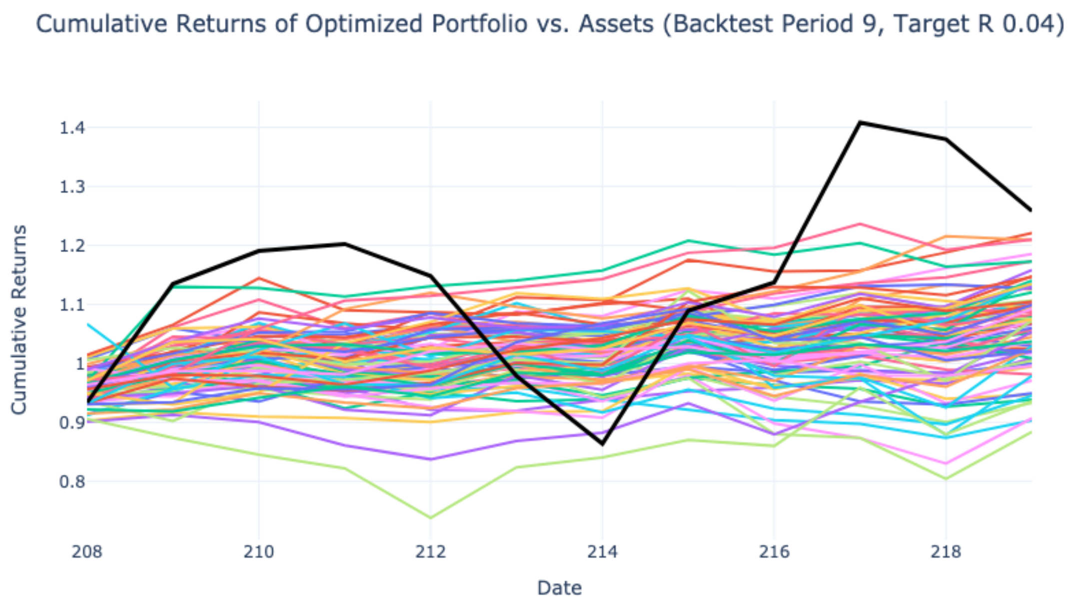


Figure 4: Cumulative Returns of Optimized Portfolio vs. Assets (Backtest Period 9, Target R 0.04)

with Figure 3, this Figure is a product of selection bias.

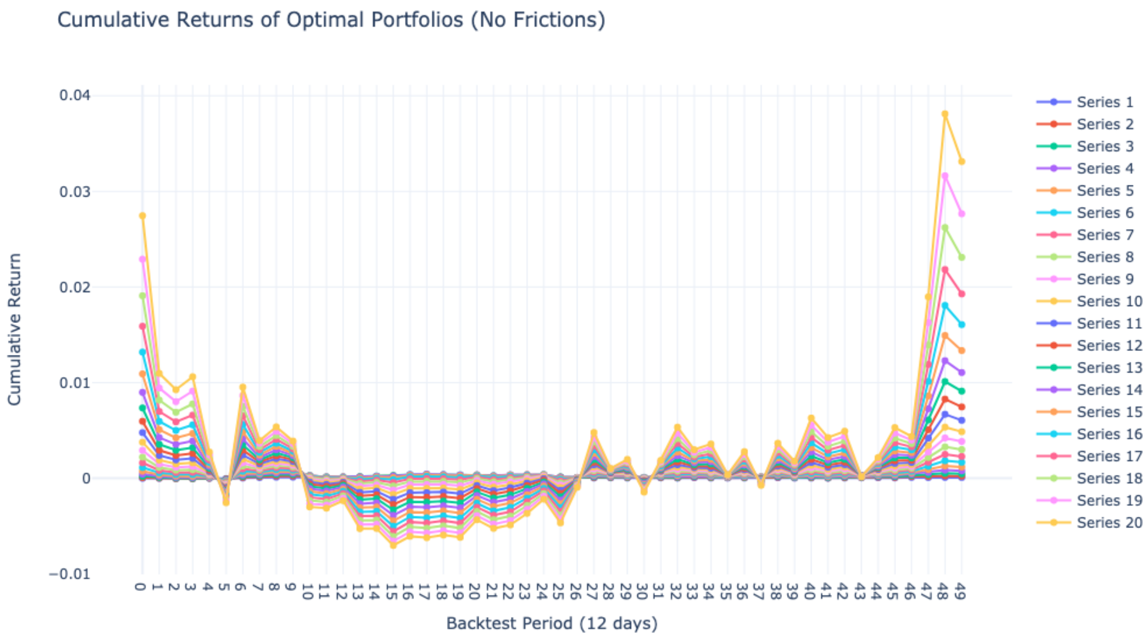


Figure 5: Cumulative Returns of Optimal Portfolios (No Frictions)

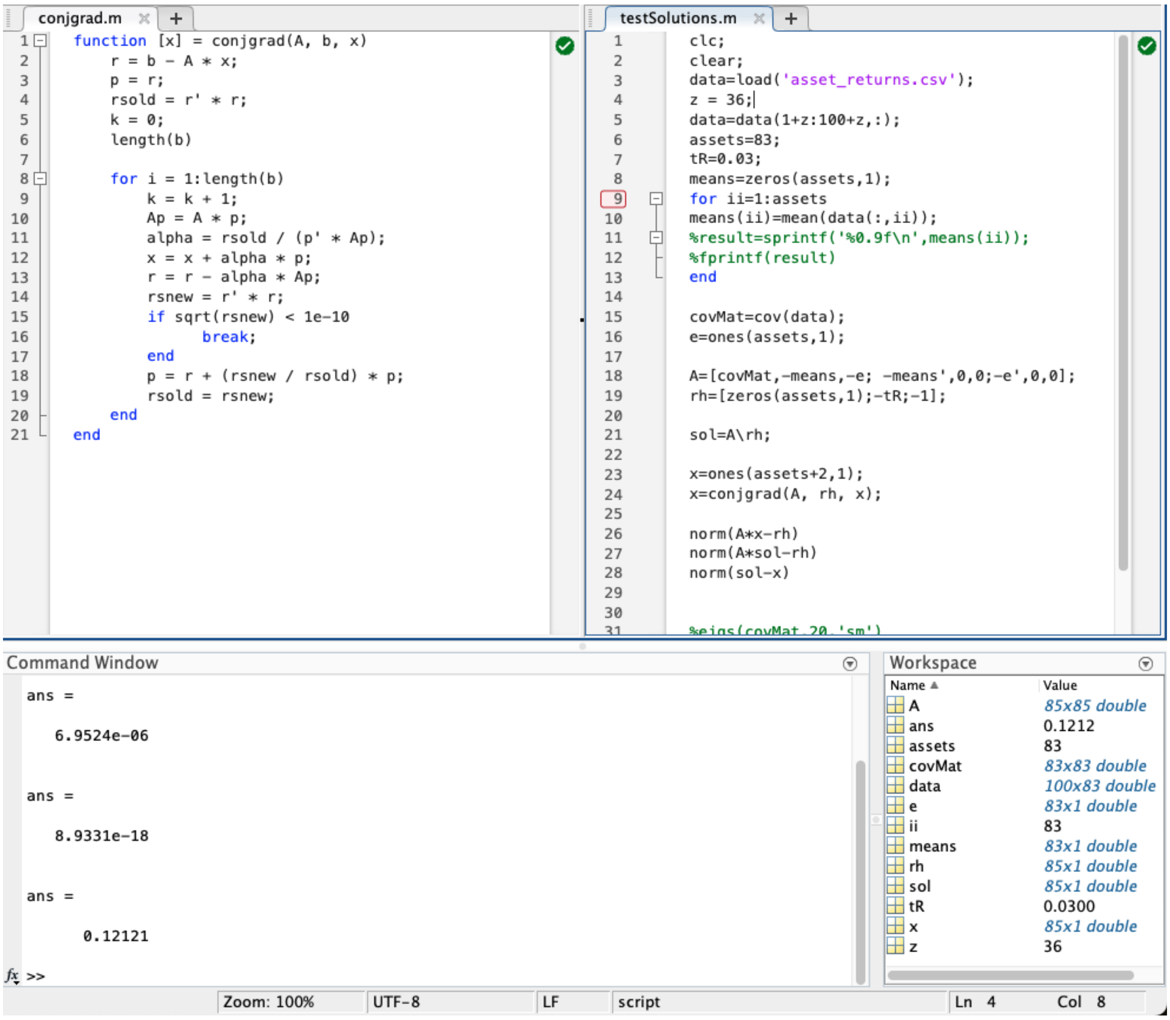


Figure 6: Matlab