Snakes on a Plane

Shrinking the Sample Covariance Matrix: A Pythonic Approach

|  |  |
| --- | --- |
| Jason Strimpel | For Dr. B. Boonstra |
| Illinois Institute of Technology | Independent Study, MSF 597 |

Introduction

When estimating the sample covariance matrix of stock returns as the input to a mean-variance optimization problem and the number of stocks under consideration is large relative to the number of historical return observations, the most extreme values in the sample covariance matrix have a lot of error. The mean-variance optimization routines will place large bets on the extreme value which is unreliable because these extreme values are largely error.

Ledoit and Wolf show that following the methods described in their paper, “Honey, I Shrunk the Sample Covariance Matrix” (http://papers.ssrn.com/sol3/papers.cfm?abstract\_id=433840), tracking error is reduced relative to a benchmark index and the information ratio is increased. This has obvious benefits in the market for active equity portfolio managers. While the authors implement their method in MATLAB, this experiment follows the empirical study outlined in the paper in Python which has become a compelling choice for work in quantitative finance, trading, and statistical applications.

Environment Used

Below describes the environment (including the Python modules) used for the experiment.

* Mac OSX 10.6 (http://store.apple.com/us/product/MC573Z/A?fnode=MTY1NDAzOA)
* Python 2.7 (http://www.python.org)
* Numpy 1.6.0 (http://numpy.scipy.org/)
* SciPy 0.9.0 (http://www.scipy.org/)
* Pandas 0.4.0dev (http://code.google.com/p/pandas/)
* PyTables 2.2.1 (http://www.pytables.org/moin)
* matplotlib 1.0.1 (http://matplotlib.sourceforge.net/)
* CVXOPT 1.1.3 (http://abel.ee.ucla.edu/cvxopt/)

Custom Modules, Classes, and Functions Used

The experiment requires XX classes, helper (private, or non-importable) methods, and regular methods.

* class Portfolio(portfolio, benchmark) contains all methods to calculate portfolio statistics. The constructor takes two arguments:
  + portfolio: user-defined dictionary containing the expected returns of each portfolio holding, the holding period of each holding, the share quantity held of each portfolio holding, and the maximum weight any position can have in the portfolio.
  + benchmark: user defined benchmark weights of the stocks held in the portfolio.

Portfolio class helper methods:

* Portfolio.\_get\_historic\_data(ticker, start, end) returns a pandas.Series with the weekly, open, high, low, close, adjusted close, and volume between start and end for ticker.
* Portfolio\_get\_historic\_returns(ticker, start, end, offset=1) returns a pandas.Series with the offset-period return between **start** and **end** for **ticker**.
* Portfolio.\_build\_portfolio(shares) returns a pandas.DataFrame with the share quantity held and the last traded price for each portfolio holding.

Portfolio class regular methods:

* Portfolio.get\_portfolio\_historic\_returns() returns a pandas.DataFrame with the period returns for each position in the portfolio.
* Portfolio.get\_portfolio\_historic\_position\_values(shares=None) returns a Pandas.DataFrame with the periodic position value (shares times price) for each position in the portfolio.
* Portfolio.get\_portfolio\_historic\_values(shares=None) returns a Pandas.Series with the periodic portfolio value.
* Portfolio.get\_benchmark\_weights() returns a pandas.DataFrame with the benchmark weights.
* Portfolio.get\_portfolio\_weights() returns a pandas.DataFrame with the weight of each portfolio holding as a percent of the total portfolio value.
* Portfolio.get\_active\_weights() returns a pandas.DataFrame with the active weight for each portfolio holding. Active weight is defined as:

where:

= portfolio weights

= benchmark weights

* Portfolio.get\_holding\_period\_returns() returns a pandas.DataFrame with the return of the portfolio holding between start and end.
* Portfolio.get\_expected\_stock\_returns() returns a pandas.DataFrame with the expected return of each portfolio holding.
* Portfolio.get\_active\_returns() returns a pandas.DataFrame with the active return for each portfolio holding. Active return is defined as:
* Portfolio.get\_expected\_excess\_stock\_returns() returns a pandas.DataFrame with the expected excess stock returns for each stock in the benchmark. Expected excess stock return is defined as:

where:

= expected stock returns

* Portfolio.get\_covariance\_matrix() ad
* Portfolio.get\_expected\_benchmark\_return() adf
* Portfolio.get\_benchmark\_variance() asd
* Portfolio.get\_expected\_portfolio\_return() ad
* Portfolio.get\_portfolio\_variance() ads
* Portfolio.get\_expected\_excess\_portfolio\_return() ad
* Portfolio.get\_tracking\_error\_variance() ads

Yahoo class:

* class Yahoo(ticker\_list) provides access to a myriad of stock information from Yahoo! Finance (http://finance.yahoo.com). There are many methods, only one of which is used. The constructor takes one argument:
  + ticker\_list: a list of ticker symbols.
* Yahoo.get\_LastTradePriceOnly(ticker) returns the last price of ticker returned to Yahoo! Finance from the applicable exchange.

Test Suite

Ledoit and Wolf published MATLAB code that generates the shrunken covariance matrix as well as the shrinkage intensity factor (http://www.ledoit.net/covCor.m) along with the paper, the results of which are used as the test suite to which my results are compared.

I computed the covariance matrix in Python and re-formatted the results to build a 30x30 element matrix in MATLAB. I passed the covariance matrix to the shrinkage method and returned a shrinkage intensity factor of 0.1852. The shrinkage intensity factor returned from the Python implementation was 0.1852, accurate to four decimal places.

To test the accuracy of the shrunk matrix, I computed the covariance matrix in Python and re-formatted the results to build a 30x30 element matrix in MATLAB. I passed the covariance matrix to the shrinkage method and returned the shrunk matrix and named the matrix sigma. I then computed the shrunk matrix in Python and re-formatted the results to build a 30x30 element matrix in MATLAB and named the variable python\_sigma. I then ran the following command in MATLAB to compute the sum of squared errors between the Python implementation and the authors’ MATLAB implementation:

>> sse = sum(sum((python\_sigma-sigma)^2))

sse =

2.3336e-015

I found this result acceptable.

Range of Inputs and Process

A faux-index of 30 stocks was used to mimic a value weighted index (see appendix A for symbols). Each period the benchmark weights were recalculated as value-weighted portfolio meaning the benchmark weight of each constituent stock equals the that period’s closing price divided by the sum of all closing prices of all stocks in the portfolio for that period.

Each stock was assigned an expected return of 3% (see Further Investigation). This of course is unrealistic but is functional for the project.

The authors build the expected excess returns, **by adding random noise to the realized excess returns, using a one-period lognormal model of returns (*(N,M)*) where  is a normally distributed variable with mean 0 and standard deviation 1 N is the number of stocks (30) and *M* is the number of periods. ** was assumed 0.03 and ** was assumed 0.05 (see Further Investigation). The authors then build ** in a such a way that the unconstrained annualized ex-ante information ratio (IR) is 1.5. This procedure is described in Appendix C of “Honey, I Shrunk the Sample Covariance Matrix”.

Monthly adjusted closing prices were used between 2/1990 and 12/2005 for the in-sample results and 2/2006 and 12/2008 for out of sample results.

I roughly followed the authors’ empirical study. The steps are outlined here:

* Compute the active returns for a rolling 60-period window and use those returns to compute the sample covariance matrix and the shrunken covariance matrix
* Compute the realized returns for that time period
* Feed the expected excess returns, **and either the sample covariance matrix or the shrunken covariance matrix (depending on the experiment) into the quadratic optimizer (see Further Investigation) which computes the optimal weights of each portfolio position which minimizes variance of the portfolio
* Compute the optimized portfolio return by multiplying the optimized weights by the current monthly realized return
* Because the expected excess returns are generated by normally distributed random variables, I run the experiment 50 times and compute the mean of the information ratio, mean expected return, and mean standard deviation

Analysis

Comparison of Results

Out of sample v. sample

Answers to Questions Posed

Further Investigation

The main purpose of this project was to implement a relevant experiment in Python using many of the modules available for quantitative financial researchers. The secondary purpose was to build a framework with which one has the ability to build from and enhance as time goes on. Given the main point was to show that using a “shrunk” covariance matrix reduces tracking error relative to a benchmark index and increases the information ratio, some details were left out.

* Expected return model. There are many models available to forecast expected returns for use in a mean-variance optimized portfolio. This experiment assumes these expected returns are given. However, the program was built with the ability to incorporate a forecasting model in the future.
* Optimizer. While I was able to implement an optimizer to return active weights given constraints, I did not fully understand how to use the code therefore improvements could be made to the optimization problem.
* Using CAPM for estimating ** and **.

Conclusion

Appendix A

Stocks used and their benchmark weights.

|  |  |
| --- | --- |
| **Company** | **Ticker** |
| Alcoa Inc. | AA |
| Apple Inc. | AAPL |
| American Express | AXP |
| Boeing Company | BA |
| Bank of America Corp. | BAC |
| BP p.l.c. | BP |
| Caterpillar, Inc. | CAT |
| Chevron Corporation | CVX |
| E. I. du Pont de Nemours and Company | DD |
| Walt Disney Company | DIS |
| General Electric | GE |
| Home Depot, Inc. | HD |
| Hewlett-Packard Company | HPQ |
| International Business Machine | IBM |
| Intel Corporation | INTC |
| Johnson & Johnson | JNJ |
| JP Morgan Chase & Company | JPM |
| Coca-Cola Company | KO |
| McDonald's Corporation | MCD |
| 3M Company | MMM |
| Merck & Company, Inc. | MRK |
| Microsoft Corporation | MSFT |
| Pfizer, Inc. | PFE |
| Procter & Gamble | PG |
| AT&T Inc. | T |
| Target Corporation | TGT |
| United Technologies | UTX |
| Verizon Communication | VZ |
| Wal-Mart Stores, Inc. | WMT |
| Exxon Mobil Corporation | XOM |