DDS Algorithm

Dominic Scruton & Philip Hindle

22 May 2020

Introduction

Portfolio Theory

Capital Asset Pricing Model (CAPM)

Sharpe Ratio as a Measure of Performance

The Expected return of a portfolio with k assets is simply the weighted sum of expected returns for each asset and is given by:

$$E[R_p] = \sum_{i=1}^{k} w_i r_i = \mathbf{w}^T \mathbf{r}$$

where $\mathbf{w}$ is a vector of stock weights and $\mathbf{r}$ is a vector of the expected returns for each stock. Defining these functions in matrix algebra notation emphasizes the use of vectorization within the DDS Algorithm. The variance of the expected return is then:

$$\begin{aligned} Var[E[R_p]] &= Var[\mathbf{w}^T \mathbf{r}] \\ &= \mathbf{w}^T Var[\mathbf{r}] \mathbf{w} \\ &= \mathbf{w}^T \Sigma \mathbf{w} \end{aligned}$$

Generally use simple returns when we have many assets. The portfolios Sharpe Ratio is then given as:

$$Sharpe = \frac{E[R_p] - r_f}{SD[R_p]}$$

The Sharpe ratio specifies the Additional expected return for a given level of risk. A higher Sharpe Ratio suggest a more optimal portfolio.

Methodology

Optimization Problem

Optimization is a common theme that underpins the fitting of Statistical models. From the maximization of the log-likelihood function under Linear Models, to the minimization of errors under Neural Network

classifiers, optimization is fundamentally important to obtaining objective solutions to many problems in the world of Data Science and Statistical research.

The general Optimization problem can be defined as follows:

$$x^* = argmin f(x; y)$$
$$x \in \Omega$$

In other words, one would like to obtain optimal values, x^*, by minimizing (or maximizing) the value of an objective function, whose value depend both on arguments x and auxiliary data, y. In some cases, it might not be possible to solve this maximization problem analytically. Analytically, the optimum can be found by differentiation, $f'(x^*) = 0$. However, in some cases there is no analytic solution, or if there is it can be difficult to derive.

In this case, the objective function, f, is the Sharpe Ratio, which one

- discuss the problems- sometimes can't differentiate, other times it is not computationally feasible to find exact answers, when we have large amounts of data and lots of stocks.

In this case, one would like to maximize the Sharpe ratio (f) such that the sum of weights in the portfolio equals one $w_i \in [0, 1], \sum_{i=1}^{K} w_i = 1$. This optimization problem can be expressed as follows:

$$Max\left(\frac{E[r_p] - r_f}{Std[E[r_p]]}\right) s.t. \sum_{i=1}^{K} w_i = 1$$

Or in vector notation:

$$Max\left(\frac{\mathbf{w^T r}}{\sqrt{\mathbf{w}^T \mathbf{\Sigma w}}}\right) s.t. \|\mathbf{w}\| = 1$$

Theoretical Solution (i.e. consider the actual mathematical result)

One may question why such effort has been expended to create the various optimization algorithms in this package and report. Whilst the exact solution is easily obtained for smaller portfolios of 'well-behaved' stocks, such a method can become computationally burdensome when the potential number of stocks is large and/or when the covariance matrix is (close to) non-invertible. In this case an exact solution may become very slow to calculate or might in fact be analytically intractable.

Discretized Dynamical System (Deterministic Grid Search)

- One of the advantages is that we can set the level of granularity here. i.e. a weight of 12.4 is effectively the same as 12.37838383838, as the qualitative results are effectively the same and in reality, one tends to . Really inefficient- needlessly searches the entire area and not at all granular- becomes extremely computationally inefficient for large numbers of stocks.

An appropriate Optimization technique should have the following properties:

1) Computationally Efficient, even for large datasets containing many stocks

2)

Gauss-Newton Method

Recall, the problem of maximizing the Sharpe Ratio is equivalent to finding weights, $\mathbf{w}^*$, such that $f'(\mathbf{w}^*) = 0$ and

$$w^* = argmax f(w; y)$$
$$w \in [0, 1]$$

Any method used to find the roots of a function can therefore also be used to optimize the function, by finding the roots of its first derivative. The one-dimensional version of Newton's Method, derived from its Taylor Expansion is expressed as follows:

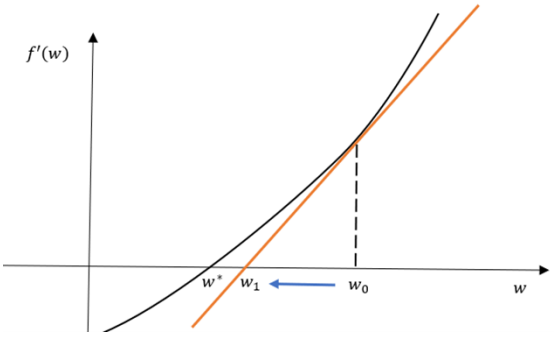
$$f'(\mathbf{w}^*) \approx f'(\mathbf{w}) + f''(\mathbf{x})(\mathbf{w} - \mathbf{w}^*)$$

, resulting in optimal weights approximated by:

$$\mathbf{w}^* \approx \mathbf{w} - \frac{f'(\mathbf{w})}{f''(\mathbf{w})}$$

One must note in this equation that $f''(\mathbf{w})$ is the derivate of the derivative of the Sharpe Ratio, whose roots we wish to identify, thus represents the rate of change of the derivative. The derivatives can either be calculated analytically or by numerical integration. A greater derivative implies the function is farther from a root. In the graph below, the algorithm is initialized at weights $\mathbf{w}_0$. One wishes to find the weights, w^* that correspond to $f'(\mathbf{w}^*) = 0$.

A further problem of Gauss-Newton Optimization is of convergence problems- the algorithm cen become circular or near circular, as it consistently over and then undershoots the optima, without getting closer as the number of iterations increases.



{width = 150%}

This method requires that the objective function is twice-differentiable and that the first and second derivatives are calculable.

Stochastic Grid Search

Simulated Aneeling

The problem we face with simulated aneeling, in comparison to its standard applications, is that the weights are bounded to lie in the interval [0, 1] and must sum to one. Therefore, we need to decide whether to use normal distributions and truncate any weights to lie in [0, 1] or use a distribution with support [0, 1] (e.g. Beta distribution).

A problem with using the normal distribution to simulate weights is that stocks with low weights may have biased estimated weights (estimated too high), because simulations yielding a negative weight are rounded up to zero. However, this stock will then have zero weight and if this is correct, the weights will stay zero throughout the rest of the simulation. Of course, several other distributions could be used, such as the lognormal, which have a support of positive real numbers $(x \in \mathbb{R}), x > 0$.

We effectively use a truncated normal distribution and in this case this is very useful, because one can reach weights of zero with high probability for stocks that provide little contribution to the Sharpe ratio (and stocks we would therefore not wish to include in a portfolio constructed according to the Sharpe ratio). Therefore, the properties of the truncated normal distrbution is that it ables one to find 'corner solutions' and can do so very efficiently for stocks that clearly should not be incorporated into a portfolio.

Discuss the Properties we want form a simulation function - Want values in interval [0, 1]. - Also want non-zero probability that simulated weights can be exactly 0, thus eliminating surplus stocks from the portfolio. -Want a standard deviation that is able to 'cool off' and decrease over time.

Other Optimization Methods

Bisection Method

- Identify an interval (a, b), where the gradient switches sign. One then computes the gradient at the midpoint, $c = \frac{a+b}{2}$ and then discards the vector of weights, a or b, for which the sign of the gradient at that point is the same as the sign of the gradient at point c. This method is simple to implement and can be easily extended to problems in an arbitrarily high number of dimensions, however typically requires many function evaluations (inefficient) and can become stuck in local and not global optima.

There exist a plethora of possible methods to solve the optimization problem identified in this report. The two general contrasting methods discussed are Stochastic and Deterministic forms of optimization.

Discuss the Advantages and disadvantages of the two methods- Deterministic and stochastic optimization methodology and also of the methods in general.

Discuss specifically why I chose to use the 3 methods that I did.

The DDS Algorithm (R Package)

Pre-Processing Function

DDS Algorithm

- Discuss how the expected returns and then the portfolio standard deviation are ANNUALIZED- see Python for Finance course

Heat Maps and Plots

Testing

To build robust software one needs to approach the problem of testing rigorously.

Unit Testing

Unit testing helps to create robust code. This ensures the code is robust to new versions of R, as well as enabling users to utilize the function correctly. This ensures the function generates the expected output and can deal with arguments of different structure, flagging problems if the structure of input arguments is not appropriate (boundary testing). This ensure code is robust and results are reproducible.

Unit testing tests functional units. For each created function, one should create further code testing whether the function works as intended. For unit testing, we typically start with function arguments such that we know the answer and then test whether the function returns the correct answer (Mailund, 2017). In this case, the problem is more complex and it is generally not possible (or not worth the time) to manually calculate the Sharpe ratio for a given set of stocks and weights. (Could either work backwards from a given Sharpe Ratio or use validated results online)

One can automate the process of testing functions by creating functions that return error messages if the results are not correct. Doing this can save time, with testing functions called whenever changes are made to each function.

In this case, because the function is not deterministic, there will be variation in the results of each function, hence a level of tolerance is set. One can also use random numbers in tests, simulating artificial data. This can often be appropriate, enabling one to test functions with more 'unusual' data, to ensure functions remain valid for outlying and extreme parameter values. If the function is not deterministic (which in our case it isn't- we have stochastic functions, all of which are approximations to the specific value).

List an example of the main data and functions used to test each function.

Boundary Testing

Other potential pitfalls of programming in R include Scoping. In R Scoping is *lexical*, so if a variable is used in a function but not defined either as an arguments or part of the function, R will start searching outward of the function for values in the global environment (Mailund, 2017) (unlike compiled languages, such as C++ and JAVA). Therefore, incorrect function arguments or values might be called that have already been created in the global environment. To avoid this, default arguments, warning statements and stop functions are used to ensure a robust function that is not prone to user error.

Boundary tests are useful in dealing with this problem. Provide a list/table of the boundary tests used on the different functions.

Table 1- Example Boundary Tests

Test	Function
Check Arguments of correct data structure	ALL
Check data specified as daily returns	DDS
Check Starting Weights Sum to 1	Simulated Aneeling

Simulation

- So far we have tested the function on 3 stocks, so we can conclude that syntactically there are no major issues. However, to assess the performance of the different simulation methods, one really needs to run the functions with a far larger number of stocks. As the number of stocks gets larger, 'edges' of the search space are less likely to be searched for random search. Furthermore, the optimzation function can get stuck in local maxima, which can be problematic.
- Need to be able to simulate data such that we start with the Sharpe ratio and work backwards, to ensure that we know the exact value of the maximum Sharpe ratio and corresponding weights, in order

to assess the performance of the different optimization methods.

Assessing Computational Efficiency

- discuss the computational efficiency of the different optimization methods- measure using the system.time function, for example.
- Also discuss how I improved the computational speed of the function using Parallelization and profiling.

Profiling is the process of analyzing the speed of execution of different parts of our code (Mailund, 2017). In particular, we want to identify inefficient and slow chunks of code and then act to improve their speed. Figuring out where these bottlenecks are requires *Profiling*. We use the *profvis* package for profiling in R and the . If we do find areas of the code which can be improved, we can improve the computational efficient of either the algorithms we run or the data structures we use.

The *microbenchmark* package enables one to compare different expressions. The microbenchmark() function runs a sequence of expressions several times and computes statistics on the execution time in units down to nanoseconds.

Simple Ways to improve computational Efficiency

- Discuss this both in terms of R and other programming languages in general.
- Discuss the measures we have used here

1) Vectorization
2) Pre-defining the dimensions of a vector/ matrix/ dataframe
3) Switching to C++

-We can also make use of the process of parallelization when more than one simulation of each algorithm is required, enabling multiple runs of the algorithm to run at once, by utilizing extra 'cores' of ones computer.

C++ enables one to program at a lower level, giving us greater control over the computer, enabling one to greatly optimize function performance. The Rcpp packages makes integrating R and C++ straightforward. However, this is beyond the scope of this paper and discussion.

The process of vectorization can be used

- Include a table for simulation speeds and results- also discuss computational complexity and efficiency

Example Portfolio

We test and simulate using stock returns from 1 January 2010 to 1 December 2020 inclusive, for Delta Airlines, Walmart and Microsoft stock in a portfolio (could add many more)

Conclusion

- Discuss the wider applicability of these methods- in Machine Learning, when we have many hyperparameters to tune, rather than using grid search, we can use Gauss- Newton and Simulated Aneeling to massively reduce computationally time and search a far greater space of hyperparameter values- huge applicability in all fields of statistics and machine learning.
- Completed all of "Beginning Data Science in R", except for the creation of packages (Chapter 11 and Chapter 14).

Bibliography

Alison Etheridge. 2002. “A Course in Financial Calculus”. Cambridge University Press.

Jonathan Berk, Peter DeMarzo. 2014. “Corporate Finance”. Pearson.

Geoffrey Grimmett, David Stirzaker. 2001. “Probability and Random Processes”. Oxford University Press.

Thomas Mailund. 2017. “Beginning Data Science in R: Data Analysis, Visualization and Modelling for the Data Scientist”. Apress.

Geof Givens, Jennifer Hoeting. 2013. “Computational Statistics”. Wiley and Sons.

Paulo Cortez. 2014. “Modern Optimization with R”. Springer.

RStudio Team (2016). RStudio: Integrated Development for R. RStudio, Inc., Boston, MA URL http://www.rstudio.com/.

Appendix

Python Code

- Attach the equivalent Python code to the above problem, however, no ‘package’ has been created-BUT could look into this.

R code

- add any extra pieces of R code here