Heuristic Methods in Finance

Enrico Schumann and David Ardia [1]

Heuristic optimization methods and their application to finance are discussed. Two illustrations of these methods are presented: the selection of assets in a portfolio and the estimation of a complicated econometric model.

Heuristic methods in Finance

Models in finance

Finance is, at its very heart, all about optimization. In financial theory, decision makers are optimizers: households maximize their utility, firms maximize profits or minimize costs. In more applied work, we may look for portfolios that best match our preferences, or for trading rules that find the ideal point in time for buying or selling an asset. And of course, the ubiquitous estimation or calibration of model parameters is nothing but optimization.

In this note we will describe a type of numerical techniques, so-called heuristics, that can be used to solve optimization models. An optimization model consists of an objective function and possibly a number of constraints, i.e., the model is a precise, mathematical description of a given problem. But the process of financial modeling comprises two stages. We start with an actual problem – such as "how to invest?" – and translate this problem into a model; then we move from the model to its numerical solution. We will be concerned with the second stage. Yet we cannot overemphasize the importance of the first stage. It may be interesting to work on a challenging optimization model, but if the model is not useful than neither is its solution.

It turns out that many financial models are difficult to solve. For combinatorial models it is the size of the search space that causes trouble. Such problems typically have an exact solution method – write down all possible solutions and pick the best one – but this approach is almost never feasible for realistic problem sizes. For continuous problems, issues arise when the objective function is not smooth (e.g., has discontinuities or is noisy), or there are many local optima. Even in the continuous case we could attempt complete enumeration by discretizing the domain of the objective function and running a grid search. But again this approach is not feasible in practice once the dimensionality of the model grows.

Researchers and operators in finance often go a long way to make models tractable, that is, to formulate them such that they can compute the quantities of interest either in closed form or with the computational tools that are at hand. When it comes to optimization, models are often shaped such that they can be solved with "classical" optimization techniques like linear and quadratic programming. But this comes at a price: we have to construct the model in such a way that it fulfills the requirements of the particular method. For instance, we may need to choose a quadratic objective function, or approximate integers with real numbers. To paraphrase John Tukey, when we solve such a model we get a precise answer but it becomes more difficult to say if we have asked the right question.

An alternative strategy is the use of heuristic optimization techniques, or heuristics for short. Heuristics aim at providing good and fast approximations to optimal solutions; to stay with Tukey's famous statement, heuristics may be described as seeking approximate answers to the right questions. (In theory, the solution of a model is the optimum; it is not necessary to speak of optimal solutions. But practically a solution is rather the result that we get from a piece of software, so it is meaningful to distinguish between good and bad solutions.) Optimization heuristics are often very simple, easy to implement and to use; there are essentially no constraints on the model formulation; and any changes to the model are quickly implemented. But of course, there must be a downside: heuristics do not provide the exact solution of the model but only a stochastic approximation. Yet such an approximation may still be better than a poor deterministic solution or no solution at all. If a model can be solved with a classical method, it is no use to try a heuristic. The advantage comes when classical

[1]The views expressed in this paper are the sole responsibility of the authors and do not necessarily reflect those of VIP Value Investment Professionals AG, aeris CAPITAL AG or any of their affiliates. Any remaining errors or shortcomings are the authors' responsibility.

methods cannot solve the given model, i.e., when a model is difficult. Such models are far more common in finance that is sometimes thought.

What is a heuristic?

The aim in optimization is to

$$\underset{x}{\text{minimize}}\, f(x, \text{data})$$

with f a scalar-valued function, and x a vector of decision variables. To maximize a function f, we minimize $-f$. In most cases, this optimization problem will be constrained. Optimization heuristics are a class of numerical methods that can solve such problems. Well-known examples are Simulated Annealing, Genetic Algorithms (or, more generally, Evolutionary Algorithms) or Tabu Search. It is hard to give a general definition of what constitutes a heuristic. Typically, the term is characterized through several criteria such as the following (e.g., Zanakis and Evans [12], Barr et al. [2]): (i) the method should give a "good" stochastic approximation of the true optimum, with "goodness" measured by computing time or solution quality, (ii) the method should be robust to changes in the given problem, in particular the problem size, (iii) the technique should be easy to implement, and (iv) implementing and using the technique should not require any subjective elements. Of course, such a definition is not unambiguous, and even in the optimization literature the term is used with different meanings.

How do heuristics work?

Very roughly, we can divide heuristics into iterative search methods and constructive methods. Constructive methods start with an empty solution and then build a solution in a stepwise manner by adding components until a solution is completed. An example: in a Traveling Salesman Problem we are given a set of cities and the distances between them. The aim is to find the route of minimal length such that each city is visited once. We could start with one city and then add the remaining cities one at a time (e.g., always choosing the nearest city) until a complete tour is created. The procedure terminates once we have found one complete solution.

For iterative search methods, we repeatedly change an existing complete solution to obtain a new solution. Such methods are far more relevant in finance, so we will concentrate on them. To describe an iterative search method, we need to specify (i) how we generate new solutions from existing solutions, (ii) when to accept such a new solution, and (iii) when to stop the search. These three decisions define a particular method; in fact, they are the building blocks of many optimization methods, not just of heuristics. As an example, think of a steepest descent method. Suppose we have a current (or initial) solution x^{c} and want to find a new solution x^{n}. Then the rules could be as follows:

(i) We estimate the slope (the gradient) of f at x^{c} which gives us the search direction. The new solution x^{n} is then $x^{\mathrm{c}} - \text{step size} \cdot \nabla f(x^{\mathrm{c}})$.

(ii) If $f(x^{\mathrm{n}}) < f(x^{\mathrm{c}})$ we accept x^{n}, i.e., we replace x^{c} by x^{n}.

(iii) We stop if no further improvements in f can be found, or if we reach a maximum number of function evaluations.

Problems will mostly occur with steps (i) and (ii). There are models in which the gradient does not exist, or cannot be computed meaningfully (e.g., when the objective function is not smooth). Hence we may need other approaches to compute a search direction. The acceptance-criterion for a steepest descent is strict: if there is no improvement, a candidate solution is not accepted. But if the objective function has several minima, this means we will never be able to move away from a local minimum, even if it is not the globally best one.

Heuristics follow the same basic pattern (i)–(iii), but they have different rules that are better suited for problems with noisy objective functions or multiple minima. In fact, almost all heuristics use one or both of the following principles.

Trust your luck Classical methods are deterministic: given a starting value, they will always lead to the same solution. Heuristics make deliberate use of randomness. New solutions may be created by randomly changing old solutions, or we may accept new solutions only with a given probability.

Don't be greedy When we compute new candidate solutions in the steepest descent method, we choose a (locally) optimal search direction (it is steepest descent after all). Many heuristics put up with "good" search directions, in many cases even random directions. Also, heuristics generally do not enforce continuous improvements; inferior solutions may be accepted. This is inefficient for a well-behaved problem with a single optimum, but it allows these methods to move away from local minima.

As a concrete example, we look at Threshold Accepting (a variant of Simulated Annealing).

(i) We randomly choose an x^{n} close to x^{c}. For instance, when we estimate the parameter values of a statistical model, we could randomly pick one of the parameters and perturb it by adding a bit of noise.

(ii) If $f(x^{\mathrm{n}}) < f(x^{\mathrm{c}})$ we accept x^{n}, as before. But if $f(x^{\mathrm{n}}) > f(x^{\mathrm{c}})$, we also accept it as long as $f(x^{\mathrm{n}}) - f(x^{\mathrm{c}})$ is smaller than a fixed threshold (which explains the method's name), i.e., we accept a new solution that is worse than its predecessor, as long as it is not too much worse. Thus, we can think of Threshold Accepting as a biased random walk. (Simulated Annealing works the same, but we would accept an inferior solution with a certain probability.)

(iii) We stop, say, after a fixed number iterations.

Stochastic solutions

Almost all heuristics are stochastic algorithms. Running the same technique twice, even with the same starting values, will typically result in different solutions. Thus, we can treat the result (i.e., the decision variables x and the associated objective function value) of a optimization heuristic as a random variable with some distribution D. We do not know what D looks like, but there is a simple way to find out for a given problem: we run a reasonably large number of restarts, for each restart we store the results, and finally we compute the empirical distribution function of these results as an estimate for D. For a given problem (often problem class), the shape of D will depend on the chosen method. Some techniques will be more appropriate than others and give less variable and on average better results. And D will often depend on the settings of the method, most importantly the number of iterations – the search time – that we allow for.

Unlike classic optimization techniques, heuristics can escape from local minima. Intuitively then, if we let the algorithm search for longer, we can hope to find better solutions. Thus the shape of D is strongly influenced by the amount of computational resources spent (often measured by the number of objective function evaluations). For minimization problems, when we increase computational resources, the mass of D will move to the left, and the distribution will become less variable. Ideally, when we let the computing time grow ever longer, D should degenerate into a single point, the global minimum. Unfortunately, it's never possible to ensure this practically.

Illustrations

Asset selection with Local Search

We can make these ideas more concrete through an example, taken from Gilli et al. [5]; sample code is given in the book. Suppose we have a universe of 500 assets (for example, mutual funds), completely described by a given variance–covariance matrix, and we are asked to find an equal-weight portfolio with minimal variance under the constraints that we have only between K_{inf} and K_{sup} assets in the portfolio. This is a combinatorial problem, and here are several strategies to obtain a solution.

(1) Write down all portfolios with feasible cardinality, compute the variance of each portfolio, and pick the one with the lowest variance.

(2) Choose k portfolios randomly and keep the one with the lowest variance.

(3) Sort the assets by their marginal variance. Then construct an equal-weight portfolio of the K_{inf} assets with the lowest variance, then a portfolio of the $K_{\mathrm{inf}} + 1$ assets with the lowest variance, and so on to a portfolio of the K_{sup} assets with the lowest variance. Of those $K_{\mathrm{sup}} - K_{\mathrm{inf}} + 1$ portfolios, pick the one with the lowest variance.

Approach (1) is infeasible. Suppose we were to check cardinalities between 100 and 150. For 100 out of 500 alone we have 10^{107} possibilities, and that leaves us 101 out of 500, 102 out of 500, and so on. Even if we could evaluate millions of portfolios in a second it would not help. Approach (2) has the advantage that it is simple, and we can scale computational resources (increase k). That is, we can use the trade-off between available computing time and solution quality. Approach (2) can be thought of as a sample substitute for Approach (1). In Approach (3) we ignore the covariation of assets (i.e., we only look at the main diagonal of the variance–covariance matrix), but we only have to check $K_{\text{sup}} - K_{\text{inf}} + 1$ portfolios. There may be cases, however, in which we would wish to include correlation.

We set up an experiment. We create an artificial data set of 500 assets, each with a randomly assigned volatility (the square root of variance) of between 20% and 40%. Each pairwise correlation is set to 0.6. We compute "best-of-k" portfolios (i.e., Approach (2)): we sample 1000 portfolios, and only keep the best one; we also try "best-of-100 000" portfolios and, for intuition, "best-of-1" portfolios (i.e., purely random ones).

Figure 1, in its upper panel, shows the estimated cumulative distribution functions of portfolio volatilities; each curve is obtained from 500 restarts. Such an empirical distribution function is an estimate of D for the particular method. We see that completely random portfolios produce a distribution with a median of about 23.5%. (What would happen if we drew more portfolios? The shape of D would not change, since we are merely increasing our sample size. But our estimates of the tails would become more precise.) We also plot the distribution of the "best-of-1000" and "best-of-100 000" portfolios. For this latter strategy, we get a median volatility below 21%. We also add the results for Approach (3); there are no stochastics in this strategy.

Now let us try a heuristic. We use a simple Local Search. We start with a random feasible portfolio and compute its volatility. This is our current solution x^c, the best solution we have so far. We now try to improve it iteratively. In each iteration we compute a new portfolio x^n as follows. We randomly pick one asset from our universe. If this asset is already in the portfolio, we remove it; if it is not in the portfolio, we add it. Then we compute the volatility of this new portfolio. If it is lower than the old portfolio's volatility, we keep the new portfolio, i.e., x^n replaces x^c; if not, we stick with x^c. We include constraints in the simplest way: if a new portfolio has too many or too few assets, we always consider it worse than its predecessor and reject it. We run this search with 100, 1000, and 10 000 iterations. For each setting, we conduct 500 restarts; each time we register the final portfolio's volatility. Results are shown in the lower panel of Figure 1.

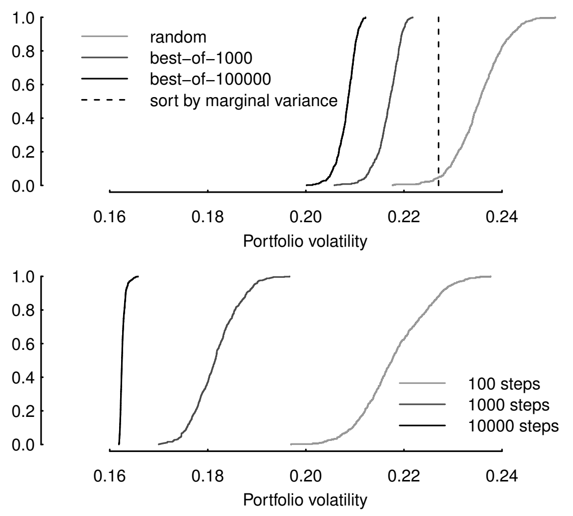

Figure 1: Upper panel: random portfolios. For example: for one restart of the "best-of-100 000" strategy, we sample 100 000 portfolios, and keep the best one. The distributions are estimated from 500 restarts. The sort-by-marginal-variance approach is deterministic, so its result is a constant. Lower panel: Local Search. Each distribution is estimated from 500 restarts.

Already with 1000 iterations we are clearly better than the "best-of-100 000" strategy (though we have used only one-hundredth of the function evaluations). With 10 000 iterations we seem to converge to a point at about 16%. A few remarks: first, we have no proof that we have found the global optimum. But we can have some confidence that we have found a good solution. Second, we can practically make the variance of D as small as we want. With more iterations (and possibly a few other refinements), we could, for all practical purposes, have the distribution "converge". But, third, in many cases we do not need to have D collapse; for financial problems a good solution is fine, given the quality of financial data [6, 7].

Econometric model fitting with Differential Evolution

Our second illustration is taken from Mullen et al. [8], who consider the estimation of a Markov-switching GARCH (MSGARCH) model. MSGARCH are econometric models used to forecast the volatility of financial time series, which is of primary importance for financial risk management. The estimation of MSGARCH models is a non-linear constrained optimization problem and is a difficult task in practice. A robust optimizer is thus required. In that regard, the authors report the best performance of the Differential Evolution (DE) algorithm compared with traditional estimation techniques.

DE is a search heuristic introduced by Storn and Price [10] and belongs to the class of evolutionary algorithms. The algorithm uses biology-inspired operations of crossover, mutation, and selection on a population in order to minimize an objective function over the course of successive generations. Its remarkable performance as a global optimization algorithm on continuous problems has been extensively explored; see, e.g., Price et al. [9].

Let NP denote the number of parameter vectors (members) $x \in \mathbb{R}^d$ in the population, where d denotes the dimension. In order to create the initial generation, NP guesses for the optimal value of the parameter vector are made, either using random values between bounds or using values given by the user. Each generation involves creation of a new population from the current population members $\{x_i \,|\, i = 1, \ldots, NP\}$, where i indexes the vectors that make up the population. This is accomplished using *differential mutation* of the population members. An initial mutant parameter vector v_i is created by choosing three members of the population, x_{i_1}, x_{i_2} and x_{i_3}, at random. Then v_i is generated as $v_i = x_{i_1} + F \cdot (x_{i_2} - x_{i_3})$, where F is a positive scale factor whose effective values are typically less than one. After the first mutation operation, mutation is continued until d mutations have been made, with a given crossover probability. The crossover probability controls the fraction of the parameter values that are copied from the mutant. Mutation is applied in this way to each member of the population. The objective function values associated with the children are then determined. If a trial vector has equal or lower objective function value than the previous vector it replaces the previous vector in the population; otherwise the previous vector remains. Note that DE uses both strategies described above to overcome local minima: it does not only keep the best solution but accepts inferior solutions, too; the method evolves a whole population of solutions in which some solutions are worse than others. And DE has a chance ingredient as it randomly chooses solutions to be mixed and mutated. For more details, see Price et al. [9] and Storn and Price [10].

We report below some results of Mullen et al. [8], who fit their model to the Swiss Market Index. For the DE optimization, the authors rely on the package **DEoptim** [1] which implements DE in the R language [3]. For comparison, the model is also estimated using standard unconstrained and constrained optimization routines available in R as well as more complex methods able to handle non-linear equality and inequality constraints. The model estimation is run 50 times for all optimization routines, where random starting values in the feasible parameter set are used when needed (using the same random starting values for the various methods). Boxplots of the objective function (i.e., the negative log-likelihood function, which must be minimized) at optimum for convergent estimations is displayed in Figure 2. We notice that standard approaches (i.e., function `optim` with all methods) perform poorly compared with the optimizers that can handle more complicated constraints (i.e., functions `constrOptim`, `constrOptim.nl` and `solnp`). DE compares favorably with the two best competitors in terms of negative log-likelihood values and is more stable over the runs.

Conclusion

In this note, we have briefly described optimization heuristics, but of course we could only scratch the surface of how these methods work and where they can be applied. After all, for most people optimization is a tool, and what matters is how this tool is applied. Heuristics offer much in this regard: they allow us to solve optimization models essentially without restrictions on the functional form of the objective function or the constraints. Thus, when it comes to evaluating, comparing, and selecting models, researchers and operators can focus more on a model's financial or empirical qualities instead of having to worry about how to handle it numerically. We have argued initially that financial modeling comprises two stages: putting an actual prob-

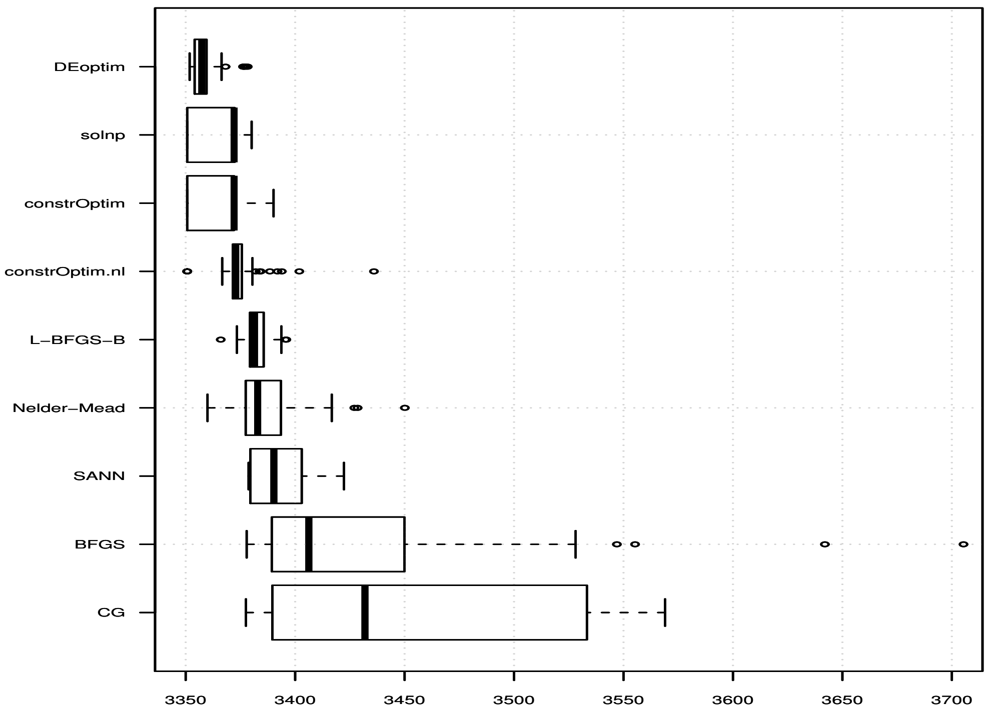

Figure 2: Boxplots of the 50 values of the objective function (i.e., the negative log-likelihood) at optimum obtained by the various optimizers available in R. Function `optim` with method `"Nelder-Mead"` (unconstrained), method `"BFGS"` (unconstrained), method `"CG"` (unconstrained), method `"L-BFGS-B"` (constrained), method `"SANN"` (unconstrained), function `constrOptim` (constrained), function `constrOptim.nl` of the package **alabama** [11], function `solnp` of the package **Rsolnp** [4], function `DEoptim` of the package **DEoptim** [1]. More details can be found in Mullen et al. [8].

lem into model form, and then solving this model. With heuristics, we become much more powerful at the second stage; it remains to use this power in the first stage.

Bibliography

[1] Ardia, D., Mullen, K. M., Peterson, B. G., Ulrich, J., 2011. **DEoptim**: Differential Evolution Optimization in R.
URL `http://CRAN.R-project.org/package=DEoptim`

[2] Barr, R. S., Golden, B. L., Kelly, J. P., Resende, M. G. C., Stewart, W. R., 1995. Designing and reporting on computational experiments with heuristic methods. Journal of Heuristics 1 (1), 9–32.

[3] R Development Core Team, 2009. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria, ISBN 3-900051-07-0.
URL `http://www.R-project.org`

[4] Ghalanos, A., Theussl, S., 2010. **Rsolnp**: General Non-linear Optimization Using Augmented Lagrange Multiplier Method.
URL `http://CRAN.R-project.org/package=Rsolnp`

[5] Gilli, M., Maringer, D., Schumann, E., 2011. Numerical Methods and Optimization in Finance. Elsevier.

[6] Gilli, M., Schumann, E., 2010. Optimal enough? Journal of Heuristics 17 (4), 373–387.

[7] Gilli, M., Schumann, E., 2010. Optimization in financial engineering – an essay on 'good' solutions and misplaced exactitude. Journal of Financial Transformation 28, 117–122.

[8] Mullen, K. M., Ardia, D., Gil, D. L., Windover, D., Cline, J., 2011. **DEoptim**: An R package for global optimization by differential evolution. Journal of Statistical Software 40 (6), 1–26.

[9] Price, K. V., Storn, R. M., Lampinen, J. A., 2006. Differential Evolution: A Practical Approach to Global Optimization. Springer-Verlag, Berlin, Germany.

[10] Storn, R., Price, K., 1997. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization 11 (4), 341–359.

[11] Varadhan, R., 2010. **alabama**: Constrained Nonlinear Optimization.
URL `http://CRAN.R-project.org/package=alabama`

[12] Zanakis, S. H., Evans, J. R., 1981. Heuristic "optimization": Why, when, and how to use it. Interfaces 11 (5), 84–91.

Enrico Schumann
VIP Value Investment Professionals AG, Switzerland
`es@vipag.com`

David Ardia
aeris CAPITAL AG, Switzerland
`da@aeris-capital.com`