Forward shooting grid for arithmetic average options

Assignment for *Applied Numerical Finance* class

Mircea Simionica
Student ID: 1814516
Bocconi University
Milan, Italy
mircea.simionica@gmail.com
mircea.simionica@studbocconi.it

Contents

1 Introduction

The forward shooting grid method is a modified binomial model that allows the valuation of different kinds of lookback options. This analysis studies the application of the forward shooting grid to Asian options. The payoff of this type of options depends on the average of the underlying asset price over some specified period of time. Asian options may look attractive to some investors because less expensive and less volatile. For this reasons they provide a cost-efficient way of hedging.

The mathematical finance literature has delved into the valuation of Asian options, which results non-trivial. Assuming stock prices follow a lognormal distribution, the geometric average presents an analytic form. However, in practice, the standard form of averaging is arithmetic. For the European case, some methods (including Monte Carlo simulation, analytic approximations through moment matching and convolution using Fast Fourier Transform) have been proposed. As is often the case, the opportunity of early exercise complicates the valuation approach even more. The forward shooting grid sets itself as an alternative to other methods.

2 Forward shooting grid method

A standard binomial method can Markovianize the problem by introducing a state variable. In our case this will be the arithmetic average of the stock price, that will be attached to every node of the tree. The complication is that the number of possible averages grows exponentially with the number of time steps. In fact, the tree for the arithmetic average is not recombining as it is for the geometric case. Hull and White [2] suggested to keep track only of a smaller subset of possible values for the arithmetic average at each node. Standard backward recursion is then used to value the option and interpolation is applied when necessary. Assuming classic notation for a binomial model, the implementation of the forward shooting grid can be sketched as follows:

- Build a standard binomial tree for the stock price. For each node (i, j) the value of the asset S will be
$$S(i, j) = S_0 u^{i-j} d^j$$
- Construct the tree of representative values for the arithmetic average. Each node (i, j) is assigned a vector of averages, whose elements are spaced between a minimum and maximum average. A way of calculating these values is considering the average generated by the path of lowest and highest prices, respectively, up to node (i, j), as illustrated in Figure 1.
- Compute the value of the option by backward recursion. Firstly, set the payoff $X(T)$ at the end nodes of the tree as

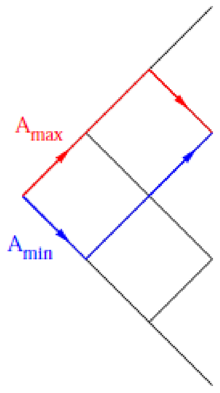

Figure 1: Paths leading to minimum and maximum average

- $X(T) = (S_{average} - K)^+$ for fixed strike call
- $X(T) = (S(T) - S_{average})^+$ for floating strike call

The recursion step can be outlined as follows. Let us consider a generic node (i, j), which will be coupled with a vector of averages $A(i, j)$. At the next time slice, the updating rule of A, corresponding to an up or down move of the stock, will provide:

$$A_u = \frac{(i+1)A_{ij} + uS_{ij}}{i+2} \quad \text{and} \quad A_d = \frac{(i+1)A_{ij} + dS_{ij}}{i+2}$$

These vectors will not coincide with the vector of representative averages computed at the second step. Thus, interpolation has to be applied in order to retrieve option values at A_u and at A_d, denoted by $V(i, j+1, A_u)$ and $V(i+1, j+1, A_d)$.

- Now standard backward recursion can be employed. As an example, for an European call

$$V(i, j) = e^{-r\Delta t}[qV(i, j+1, A_u) + (1-q)V(i+1, j+1, A_d)]$$

This way the root of the tree is reached, where there exists only one average, therefore a unique option price.

3 Numerical and financial remarks

Some remarks are in order.

- Once we calculate the minimum and maximum average for each node (i, j) we need to decide how to space values in between. Representative averages can be equally or logarithmically spaced via the following formulas:

$$A(i,j,k) = \frac{M-k}{M} A_{max}(i,j) + \frac{k}{M} A_{min}(i,j), \text{for } k = 0, 1, ..., M$$
$$A(i,j,k) = \exp\left(\frac{M-k}{M} ln(A_{max}(i,j)) + \frac{k}{M} ln(A_{min}(i,j))\right), \text{for } k = 0, 1, ..., M$$

The system described in Figure 1 is not the only way to go. Other choices of $A_{min}(i,j)$ and $A_{max}(i,j)$ are available out there, as in [1] and [4].

- Another question concers the interpolation required in the backward recursion. When the updating rule does not return an exact match (which always happens when we deal with big trees and a high number of representative averages) we can, for every element of the vector, take the closest from above, take the closest from below, or interpolate. The first two solutions do not work very well in practice, so interpolation is a must. Now, should we employ linear or quadratic interpolation? Quadratic interpolation is preferable, but may start to be effective only for a finer mesh. The code implements quadratic interpolation as well, but its behaviour is buggy along the edges of the tree so linear interpolation is used. Moreover, it might happen that some extreme values returned by the updating rule may not fall inside the vector of representative averages. In that case, extrapolation should be used. The code, however, takes the closest from above when the value falls outside from the bottom and the closest from below for the opposite case.

- Finally, the most critical part is the choice of the number of time steps in the grid and the size of the vector of representative averages. Since we are not considering the whole set of averages, but only a subset of them, the pricing procedure will not converge to exact option values unless the number of representative averages is large enough and well allocated with the size of the time step. As we depart from the root of the tree, more representative averages are necessary for convergence. The previous version of the code implemented a fixed number of representative averages, that had to be large enough compared to the number of time steps. After the class presentations and professor Battauz's suggestion, the code now attaches to every node a number of representative averages based on the position in the tree, i.e. proportionally to the index j.

- From a financial standpoint, the code is implemented for continous sampling, meaning that every node (i,j) is coupled with a vector of representative averages, which is a function of the indices i and j themselves. However, we are more interested into discrete sampling for practical reasons. My intuition is that, in this case, the vector of representative averages should be computed only at indices j where sampling happens. For the other indices, consider the generic node (i,j) where sampling is not applied. The node (i,j) can inherit the vector of one of its two predecessors, $(i-1,j-1)$ and $(i,j-1)$, or an average of them.

4 Programming remarks

The code is written in C++ and more details (dependencies, compilation and execution) can be found in Section 6. The main issue encountered in coding the forward shooting grid concerned efficient memory usage, particularly regarding the data structure that holds the binomial lattices. One of the early versions, which still attached a fixed number of representative averages to each node, ended badly due to memory bottlenecks. That version stored the lattice for S in a matrix of doubles and the lattices for the option price and the averages in a field, i.e. a matrix of vectors. This was a bad way of doing it because the space in memory is allocated on the heap for the entire matrix, even if values are not initialized. And we only need the upper triangular part of a matrix/field. A better (not claiming it is the most optimal one) way to do it is to hold the lattice for S in a sparse matrix (that does not allocate space for uninitialized values) and the other lattices in a C++ vector<> of fields. The vector<> template is an STL container that can grow dinamically, so each element will contain a field of different size, resembling an upper triangular matrix. Further improvements in memory usage came after moving away from a fixed number of representative averages, as professor Battauz suggested.

Another solution would be to not store into memory the vectors of representative averages, but to retrieve them during the backward recursion, everytime we need them. My only concern is that we gain in memory usage but loose in computational performance, since during the backward recursion, at each node, we need the vector of averages of the current and next time slice. Travelling back to the root of the tree, computations are overlapping. This brings my mind to dynamic programming and a solution would be to memoize function calls and return the cached results if already available. This point might remain a source for further investigation.

5 Numerical results

This section provides briefly some numerical results conducted using the forward shooting grid. The tests consider an European average call option, whose parameters are taken from [3]. The results of Table 1 and Figure 2 come from running the new version of the code, i.e. with vectors of representative averages of variable size. Comparing the numbers coming from the presentation (obtained runnig the old version of the code) we observe that convergence is reached without the need of pushing the number of timesteps far up. In terms of performance, the code is faster than before, since it deals with big vectors only as time travels away from the root of the tree. Moreover, in case of bigger number of time steps, it allows to end up at the final part of the tree with a large number of representative averages. For example, with 500 timesteps and a constant of proportionality of 20, the number of averages at the end nodes is 10.000. Requesting such a number for the fixed version would bring a personal computer to crash.

Number of time steps	Constant of proportionality	Option price	Time (seconds)
200	0.5	7.24680	0.450288
200	1	5.79379	1.16291
200	2	5.60352	3.14966
200	4	5.57079	10.7968
200	10	5.56145	58.977
200	15	5.56037	127.167
200	20	5.56000	219.212

Table 1: Numerical results with 200 timesteps

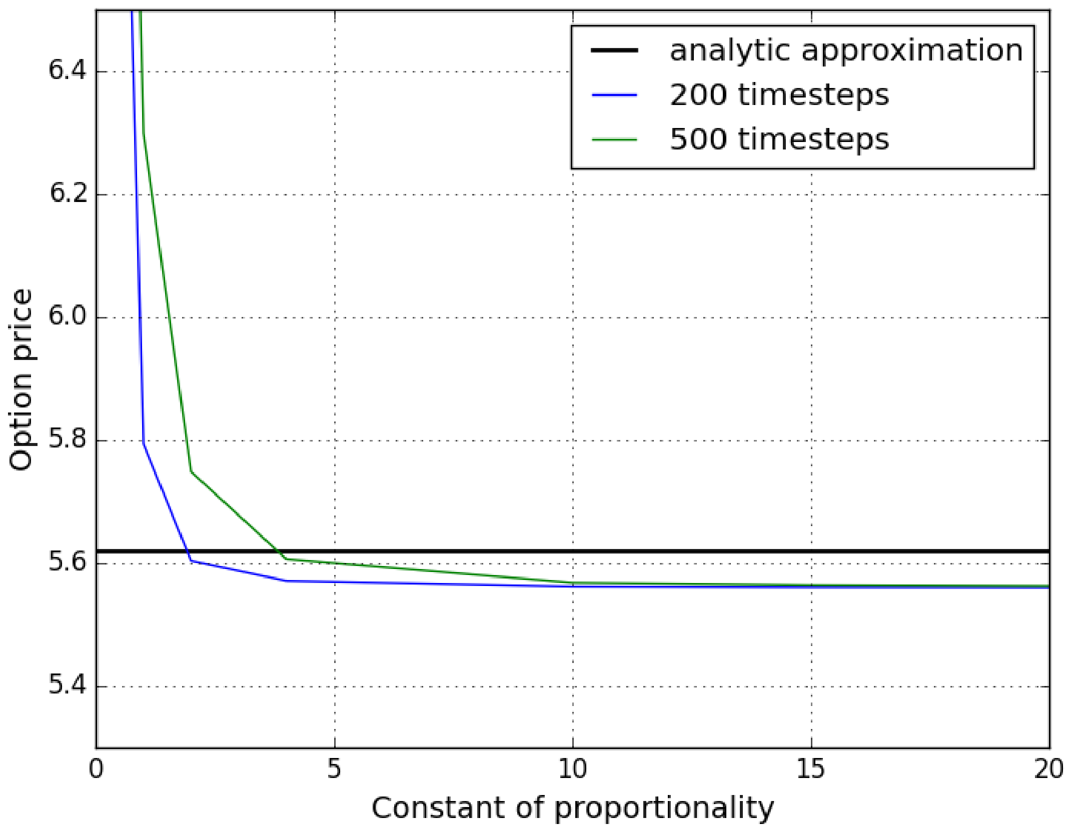

Figure 2: Convergence of the forward shooting grid

6 The code

The code is is freely available under the GNU General Public License. You need to be on a *UNIX*-type system to be able to run it. The following dependencies must be met:

- C++ compiler with C++11 support
- Armadillo: high quality C++ linear algebra library used throughout the whole code. It integrates with LAPACK and BLAS. Use Armadillo without installation and link against BLAS and LAPACK instead
- OpenBLAS: multi-threaded replacement of traditional BLAS. Recommended for significantly higher performance. Otherwise link with traditional BLAS and LAPACK.

The forward shooting grid is coded in the function *fsg_average()* and auxiliary functions (interpolation, etc.) can be found in the file *helpers.h*. Object-oriented programming is used to construct options and binomial strategies objects. This makes it easy to maintain the code and allows to plug in later different type of options (fixed, floating strike). Binomial strategies concern the computation of the parameters of the tree (up and down rate, risk neutral probability). *Option* and *binomialStrategy* are abstract classes and subsequent classes derive from them. Polymorphism is introduced in the code by taking advantage of the fact that a pointer to a derived class is type-compatible with a pointer to the base class.

To run the code you must first compile it. In the *Makefile* edit the entries regarding the location of headers and libraries. Then compile using

```
make
```

Check the file *data.dat* in the *data* directory and modify entries to your pleasure. Run the code with the following command

```
./fsg data/data.dat
```

6.1 Editing the data.dat file

In this section documentation of all the possible arguments that can be set in the *data.dat* file is provided. This is split into two sections, the essential parameters which must be set for the code to run, and optional parameters.

6.1.1 Required parameters

parameter	description	arguments
nsteps	number of timesteps	positive integer
fixedAverages	determines whether the vector of representative averages is fixed or variable	yes/no
alpha	constant of proportionality	positive integer
spot	initial value of the stock price	positive real
strike	option strike	positive real
r	risk free interest rate	positive real
maturity	option maturity (in years)	positive real
sigma	volatility of the stock	positive real
putCall	determines whether the option is a call or a put	call/put
optionType	option type: fixed or floating strike	average/asian
optionStyle	option style: European or American	E/A

6.1.2 Optional parameters

parameter	string	scheme
treeStrategy	CRR (default)	Cox-Ross-Rubinstein
	JR	Jarrow-Rudd
	JR-rn	Jarrow-Rudd risk neutral
	Tian	Tian
	CRR-drift	Cox-Ross-Rubinstein with drift
	LR	Leisen-Reimer
interpolationType	linear (default)	linear interpolation
	ceil	closest from above
	floor	closest from below
spaceType	linspace (default)	equally spaced averages
	logspace	logarithmically spaced averages

References

[1] J. Barraquand and T. Pudet. Pricing of american path-dependent contingent claims. *Mathematical Finance*, 6(17), 1996.

[2] J. Hull and A. White. Efficient procedures for valuing european and american path-dependent options. *Journal of Derivatives*, 1(1), 1993.

[3] J. C. Hull. *Options, futures and other derivatives (8th Edition)*. Prentice Hall, 2011.

[4] T. Klasse. Simple, fast and flexible pricing of asian options. *Journal of Computational Finance*, 4(3), 2001.

[5] C. Sanderson. Armadillo: an open source C++ linear algebra library for fast prototyping and computationally intensive experiments. Technical report, NICTA, 2010.