**Options Pricing in Models**

*Joseph Loss[[1]](#footnote-1), Yuchen Duan1, Daniel Liberman[[2]](#footnote-2)*

We consider options pricing in models, specifically the implementation of two algorithms listed in Feng[[3]](#footnote-3) and compared our results with those published in the original paper. Like Feng, we assume that the asset price follows a geometric process under a given and equal martingale measure. This is expressed as:

Note that is the process at and that is the asset price at .

**Algorithm 1: Normal Inverse Gaussian**

Our first algorithm is an implementation of the normal inverse Gaussian (NIG) process, which we simulated as a subordinated Brownian motion, to compute the price of a European put option contract. The NIG process is characterized by:

where is a standard Brownian motion, is an independent inverse Gaussian process and

Note that are inputs given by Feng and we discuss these on the following page.

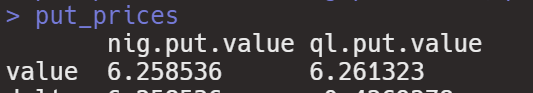
For , we simulate with the following steps:

1. Generate a standard normal random variable using the Box-Muller algorithm described on pg. 66 of Glasserman[[4]](#footnote-4). With , we compute
2. Generate a uniform random variable on . If then .   
   Otherwise, .
3. Generate a standard normal random variable Compute

To compute the price of a European vanilla put option, we use the inputs given in Section 6.1 (pg. 22) of Feng: .   
  
The price of the option at can be calculated as:

where is a process and

Using the inputs above, we computed the value of the option, $6.25836. We compared this with the Black-Scholes model for a European option in the “RQuantLib” package:



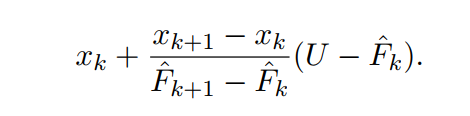
**Algorithm 2: Inverse Fourier Transform**

The second option pricing model is an implementation of the inverse transform method from tabulated probabilities which, depending on the desired accuracy required, could be multiple times faster than the normal inverse gaussian process.

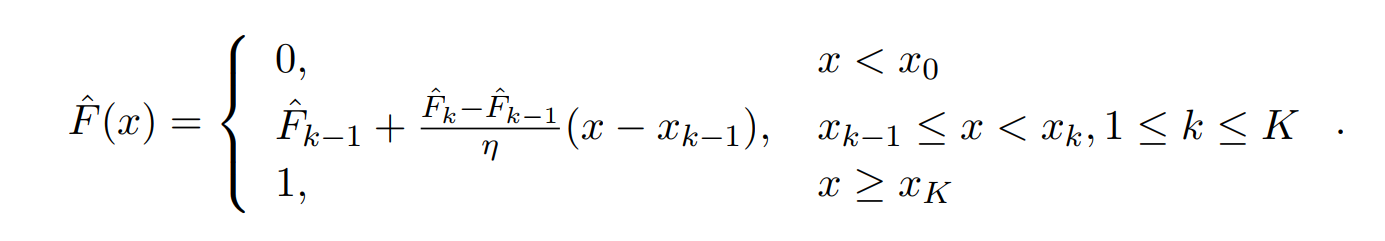
Note tInput parameters are taken from Section 6.1, pg. 22 of Feng’s paper.

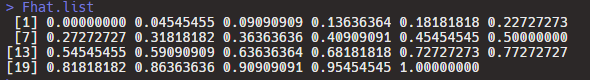
First, we begin by initializing arrays for the lists of variables and . Brute-Force-Search is used in place of the binary search originally prescribed in Section 3.1. This can be seen in the function templates for chi and Fhat.

There are two more function templates that build the foundation for this algorithm. The first, “Fhat Distribution Function”, is the direct implementation of the distribution in equation 3.12 on pg. 8 of Feng’s paper. The second, termed “Inverse Transform Function”, implements the approximation to on pg. 8 of the paper:



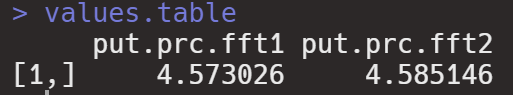
This performs the inverse transform function for each generated U between 0 and 1, utilizing brute-force search to find so that :





The Inverse Transform algorithm begins with its parameters on line 102; these parameters are taken from Section 6.1 on pg. 22 of Feng’s paper. In computing the European put price, there are two formulas used. The first method uses the formula for “European Vanilla Put Options” given in Section 5.4, pg. 19 of Feng’s paper (note the max function . The second method uses the more general,

The computed option prices are averaged over the number of MonteCarlo iterations and an example output is shown below:



Note that both prices converge to a value of $4.58 for the European put option. This is extremely close to the $4.589 value of Feng’s implementation (listed on pg. 24).

**Final Thoughts**

We are currently working on making both algorithms available in a package format, which can be downloaded, used, and improved by the community.

1. University of Illinois at Urbana-Champaign, M.S. Financial Engineering [↑](#footnote-ref-1)
2. University of Illinois at Chicago, Finance [↑](#footnote-ref-2)
3. Feng, Liming, et al. “Simulating Processes from Their Characteristic Functions and Financial Applications.” University of Illinois, 30 July 2011. [↑](#footnote-ref-3)
4. Glasserman, Paul. “Monte Carlo Methods in Financial Engineering.” Springer-Verlag, 2003. [↑](#footnote-ref-4)