4/21/2019

Joseph Loss

Algorithm 1: Normal Inverse Gaussian Process

Input Parameters:

- Taken from Section 6.1, pg. 22 of paper.
  - Note that alpha = 23 generates the correct option price of 4.58.
  - Alpha = 15 is the original parameter in the paper. This generates a put price of $6.26 and does not match Feng's Inverse-Transform price of $4.58
    - Is alpha = 15 not correct?
- No_of_simulations taken from Table 2: Algorithm 1, pg. 25 of paper

Mu and Gamma are computed using the formulas given at the top of pg. 17.

The Box-Muller Template is the implementation of pseudocode given by Paul Glasserman in *Monte Carlo Methods in Financial Engineering*, *pg. 66*.

- Note that the function uses 2*N to generate U1 and U2 independent Unif[0,1] and our N = 1.0.
  - This is because we only need 1 standard normal random variable (G1).
  - Through each iteration of the MonteCarlo simulation, the G1 returned by the BoxMuller function will be different from the last iteration. This is true for all Algorithm functions that have the loop **for (i in 1:N)**
    - If you were to change the BoxMuller N to a large number, say 1,000 and plot the result, you would see a nice bell-shaped distribution of results. However, for the NIG Algorithm, each iteration of the simulation will call the BoxMuller function to generate a new standard normal r.v. G1.

The MC algorithm, beginning on line 53, is the exact implementation of the pseudocode laid out on pg. 17 of Feng's paper:

> Step 1: Generate a standard normal random variable G1 (Box-Muller instead of Beasley-Springer-Moro) and compute Z and zeta
>
> Step 2: Generate uniform r.v. U on (0,1) and compute z_t
>
> Step 3: Generate a standard normal r.v. G2 and let $X_t = \mu t + \beta z_t + \sqrt{z_t} G_2$

From there, Xt and St are computed. Note that the for-loop was used for an optional price-path output. Remember that N=1, so for (i in 1:N) will generate just X_T and S_T (maturity).

To compute the option price, two formulas were used:

1. The standard formula for a European put option,
$$e^{-rT} * \max(0, K - S_T)$$
2. The formula for European Vanilla Options given on pg.19,
$$S_0 e^{-rT} * \max(0, \frac{K}{S_0} - \exp(\ln \frac{S_T}{S_0}))$$

This algorithm is iterated for x many simulations, and outputs the average price of the option:

```
> values.table
     nig.put.value euro_vanilla_put.value
[1,]      4.593492                4.593492
> 
```

Algorithm 2: Inverse-Transform from Tabulated Probabilities

Input Parameters:

- Taken from Section 6.1, pg. 22 of paper.

First, we begin by initializing arrays for the lists of variables χ and $\hat{F}$.

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 $F^{-1}(U)$ 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 $0 \leq k \leq K - 1$ so that $\hat{F}_k \leq U < \hat{F}_{k+1}$.

```
> Fhat.list
 [1] 0.00000000 0.04545455 0.09090909 0.13636364 0.18181818 0.22727273
 [7] 0.27272727 0.31818182 0.36363636 0.40909091 0.45454545 0.50000000
[13] 0.54545455 0.59090909 0.63636364 0.68181818 0.72727273 0.77272727
[19] 0.81818182 0.86363636 0.90909091 0.95454545 1.00000000
```

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 is $\max\left(0, \frac{strike}{S_0} - e^x\right)$. The second method uses the more general calculation of $\max(0, strike - S_0 e^{X_T})$.

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

```
> values.table
     put.prc.fft1 put.prc.fft2
[1,]     4.573026     4.585146
```

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).

As a final note, please refer to the graph below for a plot of chi.list (i.e. χ_0 to χ_K).

Plot of χ_0 to χ_K