Options Pricing in Discrete Lévy Models

Source Code and Documentation:
https://github.com/chicago-joe/Option-Pricing-via-Levy-Models

Joseph Loss

Yuchen Duan

Daniel Liberman

Background: European Vanilla Put Options[†]

NORMAL INVERSE GAUSSIAN ALGORITHM

$$S_T = S_0 * e^{X_T}$$

- NIG process is simulated through a Brownian subordination for $t > 0$:

$$X_t = \mu t + \beta z_t + \sqrt{z_t} G_2 \qquad G_2 \sim N(0,1)$$

INVERSE-TRANSFORM METHOD

$$S_T = S_0 * e^{X_T}$$

- Simulate a random variable $\mathrm{U} \sim Unif(0,1)$
- Approximate $F^{-1}(U)$ via binary search:

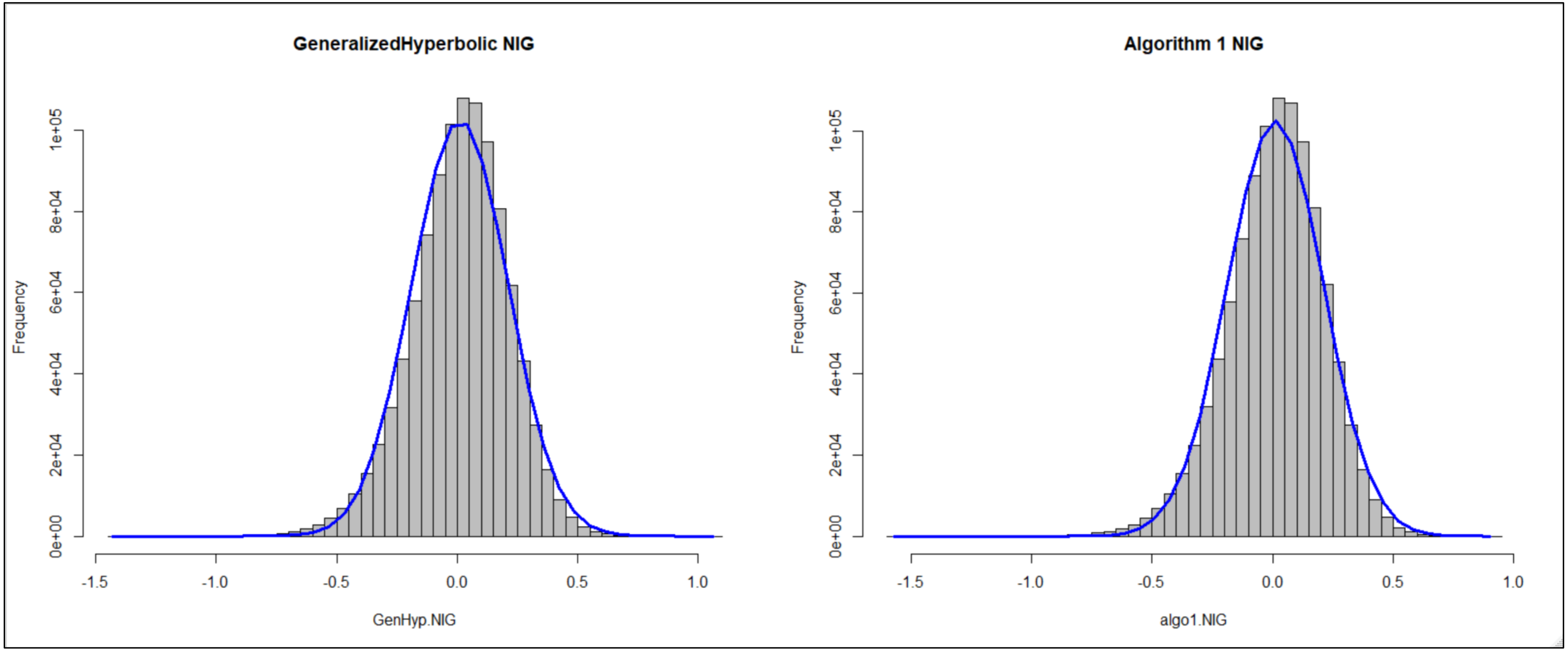
$$x_k + \frac{x_{k+1} - x_k}{\hat{F}_{k+1} - \hat{F}_k}(U - \hat{F}_k)$$

[†] *Liming Feng, et al. "Simulating Lévy Processes from Their Characteristic Functions and Financial Applications." University of Illinois, 30 July 2011*

Normal Inverse Gaussian Algorithm

```
  # Step 3: Generate StdNorm variable G2 ------------------------------------------------
  G2 <- rnorm(1,0,1)

  St <- rep(0, N)        # create empty vector for St
  Xt <- rep(0, N)        # create empty vector for Xt

  for (i in 1:N)
  {
    Xt[i] = mu * T + beta * zt[i] + sqrt(zt[i]) * G2[i]     # compute Xt = mu*t + beta*zt + sqrt(zt)*G2
    St[i] = s0 * exp(Xt[i])                                  # from pg 16. compute St = S0 * e^(Xt)
  }

  nigv[j] = Xt[N]

  stock_prc[j] = St[N]                                 # reassign variable, ie St = ST (stock value at maturity)

  put_prc[j] = exp(-r*T) * max(0, K - stock_prc[j])     # compute put price at maturity

# Section 5.4, pg.19: European Vanilla Options ------------------------------------------
  # Calculate the option value "V" using the formula given in Section 5.4.

  euro_vanilla_put[j] = s0 * exp(-r * T) * max(0, K/s0 - exp(log(St[N] / s0)))
  # note that the resulting output is identical

}
# END MonteCarlo Simulation -------------------------------------------------------------
# ---------------------------------------------------------------------------------------
euro_vanilla_put.value <- sum(euro_vanilla_put) / no_of_simulations
"European Vanilla Put Value: "
euro_vanilla_put.value
```

Normal Inverse Gaussian Distribution



Inverse-Transform Algorithm

```
# Inverse Transform Function ------------------------------------------------
# Approximation to F-1(U) using brute-force search
inverse_transform_method <- function() {
  U = runif(1,0,1)
  xk_1 = binary_search(Fhat.list, U, 1, length(Fhat.list))

  if (xk_1 < (K-1)) {                   # function returns K-1
    return(chi.list[xk_1 + 1] + (chi.list[xk_1 + 2] - chi.list[xk_1 + 1]) /
             (Fhat.list[xk_1 + 2] - Fhat.list[xk_1 + 1]) * (U - Fhat.list[xk_1 + 1]))
  } else {
    return(0)
  }
}
Xt = inverse_transform_method()

# initialize price lists
stock_prices.list <- rep(0, no_of_simulations)
put_prices.list <- rep(0, no_of_simulations)

# Inverse Transform Method 1 -----------------------------------------------
# Calculate V using Section 5.4 of Feng's Paper (pg.19)
put_prcs <- rep(0, no_of_simulations)
for (j in 1:no_of_simulations) {
  put_prcs[j] = s0 * exp(-r*T) * max(0, strike/s0 - exp(inverse_transform_method()))
}

# Method 1 Result
put_prc.InvT <- sum(put_prcs) / no_of_simulations
```

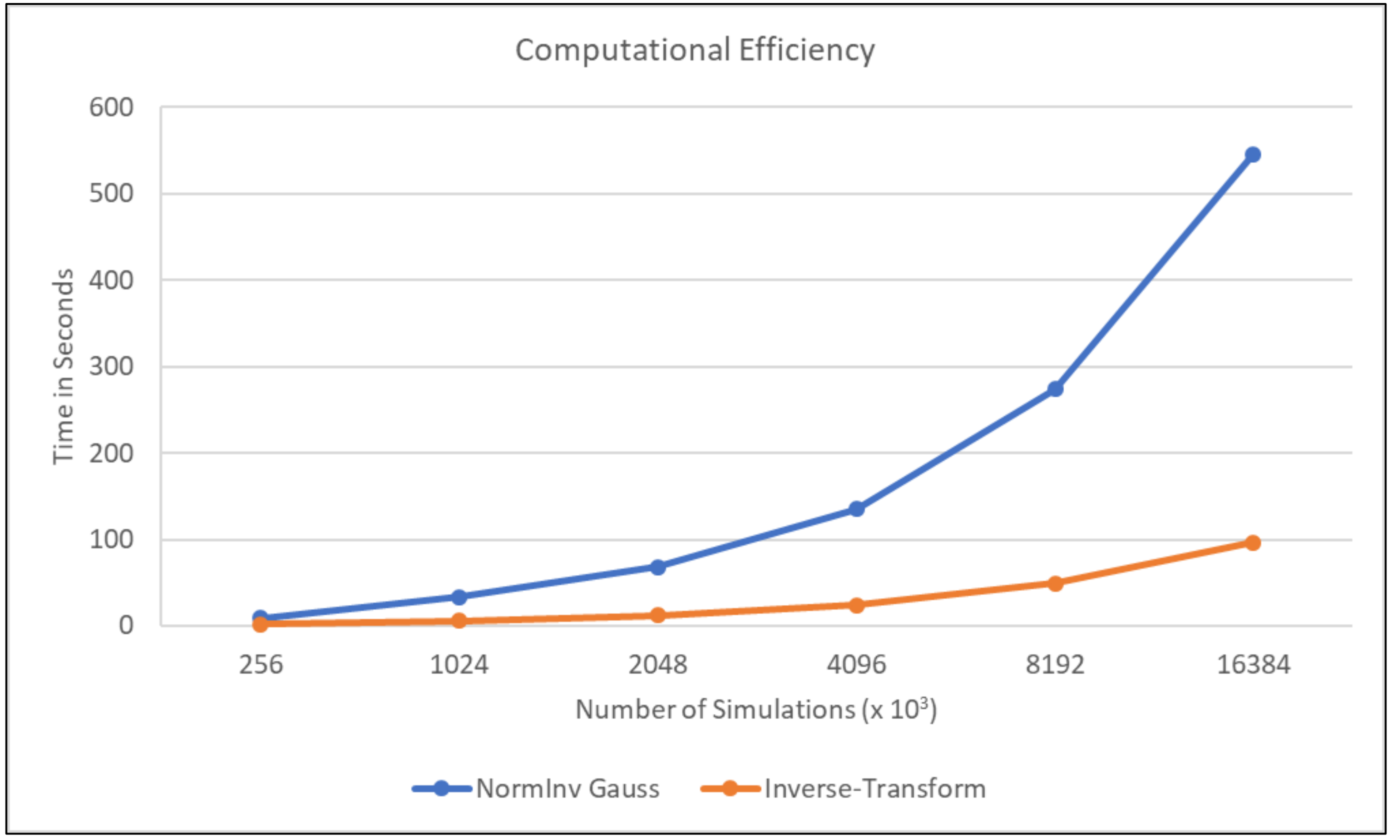
- Simulating from the distribution of $\hat{F}(x)$:

$$\begin{cases} 0, & x < x_0 \\ \hat{F}_{k-1} + \dfrac{\hat{F}_k - \hat{F}_{k-1}}{\eta}(x - x_{k-1}), & x_{k-1} < x < x_k \\ 1, & x \geq x_K \end{cases}$$

Comparison: Timing Efficiency Plots

$N\ (\times 10^3)$	**NormInv Gauss**	**Inverse-Transform**
256	8.68	1.68
1024	33.41	6.03
2048	67.98	12.02
4096	135.16	24.36
8192	274.14	49.06
16384	545.35	96.21

** computational time in seconds*



Contact Information

Joseph Loss

loss2@illinois.edu

University of Illinois

MS Financial Engineering

HIRE ME FOR SUMMER!!

Yuchen Duan

yuchend3@illinois.edu

University of Illinois

MS Financial Engineering

HIRE ME FOR SUMMER!!

Daniel Liberman

danliber92@gmail.com

University of Illinois

BS Finance

Back To School…