

# Scientific computation/Monte Carlo method

## Applied Stochastic Processes (FIN 514)

Instructor: Jaehyuk Choi

Peking University HSBC Business School, Shenzhen, China

2023-24 Module 3 (Spring 2024)

# Scientific Computation

- Most of science (including finance) problems can not be solved analytically. Even analytic/algebraic solutions still requires to be computed!
- We need problem solving methods tailored for computation.
- **Scientific computation** is a branch of (applied)mathematics/computer science to research efficient computation method.
- Scientific computation is also an important part of financial engineering/quantitative finance.

# Example: numerical root finding

- Abel (1802-1829) proved that there is no general *algebraic* solution for the roots of a quintic equation ( $a_5x^5 + \cdots + a_1x + a_0 = 0$ )
- However, we can find the root numerically:

- Bisection

$$x_{n+1} = \frac{a + x_n}{2} \quad \text{or} \quad \frac{x_n + b}{2}$$

- Newton's method

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

- Newton-Raphson method

$$x_{n+1} = x_n - f(x_n) \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}$$

- Square root:  $x = \sqrt{y}$  [Demo]

$$x_{n+1} = \frac{1}{2} \left( x_n + \frac{y}{x_n} \right)$$

# Tips and cautions

- In general, carefully translate math formulas to computer implementation
- Avoid repeated evaluation. Special functions,  $\exp$ ,  $\sin$ ,  $\cos$ ,  $\log$ , are several times expensive than that of multiplication.
- Even polynomials (Horner's rule):

$$a_0 + a_1x + \cdots + a_nx^n \quad \text{v.s.} \quad a_0 + x(a_1 + x(\cdots x(a_{n-1} + x a_n)))$$

- In computer, small difference (machine epsilon) is rounded to 0, which is unexpected and dangerous. For example,

$$f(x) = \sqrt{x+1} - \sqrt{x}$$

we get  $f(10^{16}) = 0$ . So better use the equivalent same formula,  
[Python Demo]

$$f(x) = \frac{1}{\sqrt{x+1} + \sqrt{x}}$$

# Monte-Carlo (MC) Method

- Even with computer, many problems can not be solved with deterministic (or convergent) method. So we need Monte-Carlo simulation.
- In finance, the focus is on the price of derivative, which is the expected value of stochastic payout

$$E(f(x)) \quad \text{or} \quad E(f(x_1, \dots, x_D))$$

where  $x, x_1, \dots, x_D$  are (possibly correlated)  $D$ -dimensional random numbers.

- MC is favored in a large dimensions ( $D$ ) because the cost of MC grows linearly ( $\sim D$ ) whereas the cost of the deterministic method grows exponentially ( $\sim e^D$ ), which is called **the curse of dimensionality**.
- Example: MC computation of  $\pi$  [Python Demo]

# Generating Uniform Random Number (RN)

The uniform random number between 0 and 1 is the mother of all RNs. By nature, computer **cannot** generate true RNs. In fact, true RNs are not even desirable as we want to control and reproduce the result. So we generate pseudo-RNs in the following way:

$$X_{n+1} = AX_n + B \pmod{C}, \quad U_n = X_n/C$$

$X_0$  is the **seed** of the RN generation. Using a fixed seed, you get the same result. Two popular, but outdated, choices of  $(A, B, C)$  are

- $A = 65,539 = 2^{16} + 3, \quad C = 2^{31}, \quad (B = 0)$
- $A = 16,807 = 7^5, \quad C = 2^{31} - 1, \quad (B = 0)$

The Mersenne Twister is the standard pseudo-RN generation method ( $C = 2^{19937} - 1$ ). It is implemented in modern programming languages including python.

# Generating Discrete RN

Bivariate RNs, e.g., random walk takes  $+1$  and  $-1$  with probabilities  $p$  and  $1 - p$  respectively can be generated as

$$X_k = \begin{cases} 1 & \text{if } U_k < p \\ -1 & \text{if } U_k > p \end{cases},$$

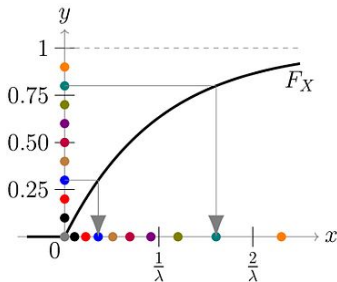
where  $U_k$  is uniform RNs. Multiple values can be similarly generated.

# Generating RN from a General Distribution

If we know the cumulative distribution function (CDF) of a distribution  $P(x)$ , the **inverse** CDF gives random numbers following the distribution,

$$X = P^{-1}(U)$$

However, probability distribution with analytically invertible CDF is rare. When inverse CDF is not available, the last resort is numerical root-finding,  $P(X) = U$  but this is computationally expensive. Finding efficient sampling method is an important area of research.





# Generating Exponential Distribution RNs

## Exponential/Poisson distribution

- Distribution for the survival time or arrival time  $T$  with intensity  $\lambda$ .
- PDF:  $f(t) = \lambda e^{-\lambda t}$ , CDF:  $F(t) = 1 - e^{-\lambda t}$
- $E(T) = 1/\lambda$ ,  $\text{Var}(T) = 1/\lambda^2$ .

The CDF function is easily invertible. The RN can be generated with

$$T_k = -\frac{1}{\lambda} \log U_k = \left( -\frac{1}{\lambda} \log(1 - U_k) \right)$$

where  $U_k$  is uniform RNs.

# Gaussian Distribution

- Given the wide usage, efficient evaluation of Gaussian distribution is critically important.
- How to evaluate CDF? Integration is very expensive!

$$N(z) = \int_{-\infty}^z n(s) ds$$

- An accurate approximation is used instead ( $x > 0$ ):

$$1 - N(x) = 1 - (a_1 t + a_2 t^2 + a_3 t^3) e^{-x^2/2}, \quad t = \frac{1}{1 + px/\sqrt{2}}$$

where  $a_1 = 0.3480242$ ,  $a_2 = -0.0958798$ ,  $a_3 = 0.7478556$ , and  $p = 0.47047$ .

# Gaussian RN (Box-Muller Method)

- Reference: Box, George EP, and Mervin E Muller. [A Note on the Generation of Random Normal Deviates](#). The Annals of Mathematical Statistics 29, no. 2 (1958): 610–11.
- Evaluation of the normal inverse CDF,  $N^{-1}$ , is expensive.
- There is a brilliant trick of generating Gaussian RN:

For two-dimensional Gaussian random numbers  $(Z_1, Z_2)$

$$\begin{aligned} P(Z_1^2 + Z_2^2 < y) &= \int_{Z_1^2 + Z_2^2 < y} \frac{1}{\sqrt{2\pi}^2} e^{-\frac{1}{2}(z_1^2 + z_2^2)} dz_1 dz_2 \\ &= \frac{2\pi}{2\pi} \int_0^{\sqrt{y}} r e^{-r^2/2} dr = 1 - e^{-y/2} \end{aligned}$$

The squared radius,  $Y = Z_1^2 + Z_2^2$ , follows exponential distribution and we know the inverse CDF is available:

$$F_Y(y) = 1 - e^{-y/2}, \quad F_Y^{-1}(u) = -2 \log(1 - u).$$

## Gaussian RN (Box-Muller Method) cont.

- This means we can accurately sample  $Y$  and  $R = \sqrt{Y}$ :

$$Y = -2 \log(1 - U_1) \quad \text{for a uniform RV, } U_1$$

$$R = \sqrt{Y} = \sqrt{-2 \log U_1} \quad (\text{symmetry: } U_1 \leftrightarrow 1 - U_1)$$

- Then, we extract two normal RNs,  $Z_1$  and  $Z_2$ , as  $x$ - and  $y$ -axis projections with a random angle  $2\pi U_2$ :

$$Z_1 = \sqrt{-2 \log U_1} \cos(2\pi U_2), \quad Z_2 = \sqrt{-2 \log U_1} \sin(2\pi U_2).$$

We get two Gaussian RNs from two uniform RNs.

# Gaussian RN (Marsaglia-Polar Method)

- Reference: Marsaglia, George, and Thomas A Bray. [A Convenient Method for Generating Normal Variables](#). SIAM Review 6, no. 3 (1964): 260–64.
- The Box-Muller method can be further improved!
- $V_{1,2} = 2U_{1,2} - 1$  so that  $V_{1,2}$  is uniform RVs between -1 and 1.
- Take if  $0 < W = V_1^2 + V_2^2 < 1$  and reject otherwise so that  $(V_1, V_2)$  is uniform random point on the unit circle.
- $W$  has a uniform distribution on  $[0, 1]$ ,

$$P(W < x) = \pi(\sqrt{x})^2 / \pi = x,$$

so  $W$  can replace  $U_1$ .

# Gaussian RN (Marsaglia-Polar Method) cont.

- Using the trigonometric properties,

$$\left( \frac{V_1}{\sqrt{W}}, \frac{V_2}{\sqrt{W}} \right) = \left( \cos(2\pi U_2), \sin(2\pi U_2) \right)$$

- Finally we obtain

$$Z_1 = \sqrt{-2 \log W} V_1 / \sqrt{W} = V_1 \sqrt{-2(\log W)/W}$$

$$Z_2 = \sqrt{-2 \log W} V_2 / \sqrt{W} = V_2 \sqrt{-2(\log W)/W}$$

- Even after *wasting* 21% of  $(V_1, V_2)$  pairs, it is still more efficient by avoiding expensive sin and cos evaluations!

# Reducing MC variance

- Anti-thetic method; ( $\pi$  by MC)
  - By symmetry,  $U$  is a uniform random number, so is  $1 - U$
  - By symmetry,  $Z$  is a Gaussian random number, so is  $-Z$
- Importance sampling method: ( $P(Z > 30)$  by MC)
  - Original probability = Modified probability  $\times$  Correction
- Low-discrepancy sequence
  - Quasi-random numbers evenly distributed over  $(0, 1)$ .
- Control Variate: for  $g(x)$  similar to  $f(x)$  and known  $E(g(X))$ ,

$$E_{CV}(f(X)) = E_{MC}(f(X)) + \left( E(g(X)) - E_{MC}(g(X)) \right)$$

[Python Demo]

# Generation of correlated Gaussian RNs ( $N = 2$ )

- We want to generate two standard Gaussian RNs,  $W_1$  and  $W_2$ , correlated by  $\rho$ ,

$$\frac{E(W_1 W_2)}{E(W_1)E(W_2)} = E(W_1 W_2) = \rho.$$

- We can achieve that by separating correlate and de-correlated parts:

$$W_1 = Z_1, \quad W_2 = \rho Z_1 + \sqrt{1 - \rho^2} Z_2,$$

where  $Z_1$  and  $Z_2$  are independent and standard RNs.

- You can easily prove that (see StoFin mid-term exam):

$$E(W_2^2) = 1 \quad \text{and} \quad E(W_1 W_2) = \rho$$

- You'll see this trick in spread option and stochastic differential equations for stochastic volatility models (Heston and SABR model)



# Covariance/correlation matrix

For  $N$ -dimensional samples,  $X = (X_1, \dots, X_N)^T$ , with mean 0 and stdev  $\sigma_k$ , the covariance matrix  $\Sigma$  is given as

$$\Sigma_{ij} = \text{Cov}(X_i, X_j) = E((X_i - \bar{X}_i)(X_j - \bar{X}_j)) \quad (\Sigma_{ij} = \sigma_i^2).$$

The correlation matrix  $R$  is similarly defined as

$$R_{ij} = \text{Corr}(X_i, X_j) = \frac{\Sigma_{ij}}{\sigma_i \sigma_j} \quad (R_{ii} = 1)$$

- If  $\{X_i\}$  are independent standard normals,  $\Sigma = I$
- Symmetric:  $\Sigma = \Sigma^T$  ( $\Sigma_{ij} = \Sigma_{ji}$ )
- Positive-definite:  $\mathbf{a}^T \Sigma \mathbf{a} \geq 0$  for any (row) vector  $\mathbf{a}$ :

$$\text{Var}(\mathbf{a}X) = \text{Var}(a_1 X_1 + \dots + a_N X_N) = \mathbf{a} \Sigma \mathbf{a}^T \geq 0$$

- For a linear combination,  $\mathbf{L}X$ ,  $\text{Cov}(\mathbf{L}X) = \mathbf{L} \Sigma \mathbf{L}^T$ .

# Generation of correlated Gaussian RNs ( $N$ in general)

- $X = (X_i)^T$  with covariance matrix  $\Sigma$ : we want to find such that  $X = LZ$  where  $Z = (Z_i)^T$  is independent standard normals.
- Then,  $L$  should satisfy the covariance matrix condition:

$$\Sigma = \text{Var}(LZ) = LIL^T = LL^T$$

Therefore,  $L$  should be a **square-root matrix** of  $\Sigma$ :  $L = \sqrt{\Sigma}$ .

- The square-matrix  $L$  is not unique. Cholesky decomposition (lower triangular matrix) is a numerically stable and fast solution. [Python Demo]
- The 2 asset case ( $\rho = \rho_{12}$ ):

$$L = \begin{pmatrix} \sigma_1 & 0 \\ \sigma_2\rho & \sigma_2\sqrt{1-\rho^2} \end{pmatrix}, \quad \Sigma = LL^T = \begin{pmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{pmatrix}$$

# MC of Brownian Motion(s)

- Standard BM:  $W_T \sim \sqrt{T}Z_1$  where  $Z_1$  is a standard normal RV.
- Arithmetic Brownian motion (Bachelier model)

- 

$$dF_t = \sigma_N dW_t \Rightarrow F_T \sim F_0 + \sigma_N \sqrt{T} Z_1$$

- From  $t = S$  to  $t = T$ :  $F_T \sim F_S + \sigma_N \sqrt{T - S} Z_1$
- For  $N$  assets:  $\mathbf{F}_T = \mathbf{F}_0 + \mathbf{L} \sqrt{T} Z$ , where  $\mathbf{L}\mathbf{L}^T = \mathbf{\Sigma}$
- Geometric Brownian motion (BSM model)

- 

$$\frac{dF_t}{F_t} = \sigma_{BS} dW_t \Rightarrow F_T \sim F_0 \exp \left( -\frac{1}{2} \sigma_{BS}^2 T + \sigma_{BS} \sqrt{T} Z_1 \right)$$

- From  $t = S$  to  $t = T$ :  
 $F_T \sim F_S \exp \left( -\frac{1}{2} \sigma_{BS}^2 (T - S) + \sigma_{BS} \sqrt{(T - S)} Z_1 \right)$
- For  $N$  assets:  $F_k(T) = F_k(0) \exp \left( -\frac{1}{2} T \Sigma_{kk} + \mathbf{L}_{k*} \sqrt{T} Z \right)$ , where  $\mathbf{L}\mathbf{L}^T = \mathbf{\Sigma}$  and  $\mathbf{L}_{k*}$  is the  $k$ -th row vector of  $\mathbf{L}$ .

- [Python Demo]