# IEOR E4703: Monte-Carlo Simulation

## Generating Random Variables and Stochastic Processes

**Martin Haugh**

Department of Industrial Engineering and Operations Research
Columbia University
Email: `martin.b.haugh@gmail.com`

## Outline

## Monte Carlo Integration

Suppose then that we want to compute

$$\theta := \int_0^1 g(x)\ dx.$$

If we cannot compute $\theta$ analytically, then we could use numerical methods.

But can also use Monte-Carlo simulation by noting that

$$\theta = \mathbb{E}[g(U)]$$

where $U \sim U(0,1)$.

Can use this to estimate $\theta$ as follows:

1. Generate $U_1, U_2, \ldots U_n \sim$ IID $U(0,1)$
2. Estimate $\theta$ with
$$\widehat{\theta}_n := \frac{g(U_1) + \ldots + g(U_n)}{n}$$

## Monte Carlo Integration

There are two reasons that explain why $\widehat{\theta}_n$ is a good estimator:

1. $\widehat{\theta}_n$ is unbiased, i.e., $\mathbb{E}[\widehat{\theta}_n] = \theta$.
2. $\widehat{\theta}_n$ is consistent, i.e., $\widehat{\theta}_n \to \theta$ as $n \to \infty$ with probability $1$
   - follows immediately from Strong Law of Large Numbers (SLLN) since $g(U_1), g(U_2), \ldots, g(U_n)$ are IID with mean $\theta$.

Monte Carlo Integration can be especially useful for estimating high-dimensional integrals. **Why?**

## An Example

Wish to estimate $\theta = \int_1^3 (x^2 + x)\ dx$ again using simulation.

Can estimate it by noting that

$$\begin{aligned} \theta &= 2\int_1^3 \frac{x^2+x}{2}\, dx \\ &= 2\mathbb{E}[X^2 + X] \end{aligned}$$

where $X \sim U(1,3)$.

So can estimate $\theta$ by:

1. Generating $n$ IID $U(0,1)$ random variables
2. Converting them (how?) to $U(1,3)$ variables, $X_1, \ldots, X_n$
3. Then taking

$$\widehat{\theta}_n := 2\sum_{i=1}^{n}(X_i^2 + X_i)/n.$$

# High-Dimensional Monte Carlo Integration

Can also apply Monte Carlo integration to more general problems.

**e.g.** Suppose we want to estimate

$$\theta := \int \int_A g(x, y) f(x, y) \; dx \, dy$$

where $f(x, y)$ is a density function on $A$.

Then observe that $\theta = \mathbb{E}[g(X, Y)]$ where $X$, $Y$ have joint density $f(x, y)$.

To estimate $\theta$ using simulation we simply generate $n$ random vectors $(X, Y)$ with joint density $f(x, y)$ and then estimate $\theta$ with

$$\widehat{\theta}_n := \frac{g(X_1, Y_1) + \; \ldots \; + g(X_n, Y_n)}{n}.$$

## Generating Univariate Random Variables

There are many methods for generating univariate random variables:

1. The inverse transform method
2. The composition method
3. The acceptance-rejection (AR) algorithm
4. Other approaches.

## The Inverse Transform Method: Discrete Random Variables

Suppose $X$ can take on $n$ distinct values, $x_1 < x_2 < \ldots < x_n$, with

$$P(X = x_i) = p_i \quad \text{for} \quad i = 1, \ldots, n.$$

Then to generate a sample value of $X$ we:

1. Generate $U$
2. Set $X = x_j$ if $\sum_{i=1}^{j-1} p_i < U \leq \sum_{i=1}^{j} p_i$.

   That is, we set $X = x_j$ if $F(x_{j-1}) < U \leq F(x_j)$.

Should be clear that this algorithm is correct!

If $n$ is **large**, then might want to search for $x_j$ more efficiently!

## Example: Generating a Geometric Random Variable

$X$ is geometric with parameter $p$ so $P(X = n) = (1-p)^{n-1}p$.

Can then generate $X$ as follows:

1. Generate $U$
2. Set $X = j$ if $\sum_{i=1}^{j-1}(1-p)^{i-1}p < U \leq \sum_{i=1}^{j}(1-p)^{i-1}p$.
   That is, $X = j$ if $1-(1-p)^{j-1} < U \leq 1-(1-p)^{j}$.

Step 2 amounts to setting $X = \text{Int}\left(\frac{\log(U)}{\log(1-p)}\right) + 1$.

**Question:** How does this compare to the coin-tossing method for generating $X$?

## Inverse Transform for Continuous Random Variables

Suppose now that $X$ is a continuous random variable.

When $X$ was discrete, we could generate a variate by first generating $U$ and then setting $X = x_j$ if $F(x_{j-1}) < U \leq F(x_j)$.

This suggests that when $X$ is continuous, we might generate $X$ as follows:

1. Generate $U$
2. Set $X = x$ if $F_x(x) = U$, i.e., set $X = F_x^{-1}(U)$.

Need to prove that this algorithm actually works! But this follows since

$$\begin{aligned} P(X \leq x) &= P(F_x^{-1}(U) \leq x) \\ &= P(U \leq F_x(x)) \\ &= F_x(x) \end{aligned}$$

as desired.

## Inverse Transform for Continuous Random Variables

This argument assumes $F_x^{-1}$ exists.

But there is no problem even when $F_x^{-1}$ does not exist. All we have to do is:

1. Generate $U$
2. Set $X = \min\{x : F_x(x) \geq U\}$.

This works for discrete and continuous random variables or mixtures of the two.

## Example: Generating an Exponential Random Variable

We wish to generate $X \sim \mathsf{Exp}(\lambda)$.

In this case $F_x(X) = 1 - e^{-\lambda x}$ so that $F_x^{-1}(u) = -\log(1-u)/\lambda$.

Can generate $X$ then by generating $U$ and setting (why?) $X = -\log(U)/\lambda$.

# Generating Order Statistics via Inverse Transform

Suppose $X$ has CDF $F_x$ and let $X_1, \ldots, X_n$ be IID $\sim X$.

Let $X_{(1)}, \ldots, X_{(n)}$ be the **ordered** sample so that

$$X_{(1)} \leq X_{(2)} \leq \ldots \leq X_{(n)}.$$

We say $X_{(i)}$ is the $i^{th}$ ordered statistic.

Several questions arise:

1. How do we generate a sample of $X_{(i)}$?
2. Can we do better?
3. Can we do even better? **Hint:** Suppose $Z \sim \text{beta}(a, b)$ on $(0, 1)$ so that

   $$f(z) = cz^{a-1}(1-z)^{b-1} \quad \text{for } 0 \leq z \leq 1$$

   where $c$ is a constant.

## Advantages / Disadvantages of Inverse Transform Method

Two principal advantages to the inverse transform method:

1. Monotonicity – have already seen how this can be useful.
2. The method is 1-to-1, i.e. one $U(0,1)$ variable produces one $X$ variable
   - can be useful for some variance reduction techniques.

Principal disadvantage is that $F_x^{-1}$ may not always be computable.

**e.g.** Suppose $X \sim \mathsf{N}(0,1)$. Then

$$F_x(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} \exp\left(\frac{-z^2}{2}\right) \, dz$$

so that we cannot even express $F_x$ in closed form.

Even if $F_x$ is available in closed form, it may not be possible to find $F_x^{-1}$ in closed form.

**e.g.** Suppose $F_x(x) = x^5(1+x)^3/8$ for $0 \leq x \leq 1$. Then cannot compute $F_x^{-1}$.

One possible solution to these problems is to find $F_x^{-1}$ **numerically**.

## The Composition Approach

Can often write

$$F_x(x) = \sum_{j=1}^{\infty} p_j F_j(x)$$

where the $F_j$'s are also CDFs, $p_j \geq 0$ for all $j$, and $\sum p_j = 1$.

Equivalently, if the densities exist then we can write

$$f_x(x) = \sum_{j=1}^{\infty} p_j f_j(x).$$

Such a representation often occurs very naturally.

**e.g.** Suppose $X \sim$ Hyperexponential$(\lambda_1, \alpha_1, \ldots, \lambda_n, \alpha_n)$ so that

$$f_x(x) = \sum_{j=1}^{n} \alpha_i \lambda_i e^{-\lambda_i x}$$

where $\lambda_i,\ \alpha_i \geq 0$, and $\sum_i^n \alpha_i = 1$.

If difficult to simulate $X$ using inverse transform then could use the composition algorithm instead.

## The Composition Algorithm

1. Generate $I$ that is distributed on the non-negative integers so that $P(I = j) = p_j$. (How do we do this?)
2. If $I = j$, then simulate $Y_j$ from $F_j$
3. Set $X = Y_j$

Claim that $X$ has the desired distribution!

**Proof.** We have

$$\begin{aligned} P(X \le x) &= \sum_{j=1}^{\infty} P(X \le x | I = j) P(I = j) \\ &= \sum_{j=1}^{\infty} P(Y_j \le x) P(I = j) \\ &= \sum_{j=1}^{\infty} F_j(x) p_j \\ &= F_x(x). \end{aligned}$$

□

## The Acceptance-Rejection Algorithm

Let $X$ be a random variable with density, $f(\cdot)$, and CDF, $F_x(\cdot)$.

Suppose it's hard to simulate a value of $X$ directly using inverse transform or composition algorithms.

Might then wish to use the acceptance-rejection algorithm.

Towards this end let $Y$ be another r.var. with density $g(\cdot)$ and suppose it's easy to simulate $Y$.

If there exists a constant $a$ such that

$$\frac{f(x)}{g(x)} \leq a \text{ for all } x$$

then can simulate a value of $X$ as follows.

## The Acceptance-Rejection Algorithm

```
generate $Y$ with PDF $g(\cdot)$
generate $U$
while $U > \frac{f(Y)}{ag(Y)}$
    generate $Y$
    generate $U$
set $X = Y$
```

Must prove the algorithm does indeed work.

So define $B$ to be event that $Y$ has been accepted in the while loop, i.e., $U \leq f(Y)/ag(Y)$.

We need to show that $P(X \leq x) = F_x(x)$

## The Acceptance-Rejection Algorithm

**Proof.** First observe

$$P(X \leq x) = P(Y \leq x | B) = \frac{P\left((Y \leq x) \ \cap \ B\right)}{P(B)}. \tag{1}$$

We can compute $P(B)$ as

$$P(B) \quad = \quad P\left(U \leq \frac{f(Y)}{ag(Y)}\right) \ = \ \frac{1}{a}$$

while the numerator in (1) satisfies

$$\begin{aligned} P\left((Y \leq x) \ \cap \ B\right) \quad &= \quad \int_{-\infty}^{\infty} P\left((Y \leq x) \ \cap B \mid Y = y\right) g(y) \ dy \\ &= \quad \int_{-\infty}^{\infty} P\left((Y \leq x) \ \cap \left(U \leq \frac{f(Y)}{ag(Y)}\right) \ \middle| \ Y = y\right) g(y) \ dy \\ &= \quad \int_{-\infty}^{x} P\left(U \leq \frac{f(y)}{ag(y)}\right) g(y) \ dy \quad \text{(why?)} \\ &= \quad \frac{F_x(x)}{a} \end{aligned}$$

Therefore $P(X \leq x) = F_x(x)$, as required. $\square$

## Example: Generating a Beta$(a, b)$ Random Variable

Suppose we wish to simulate from the Beta$(4, 3)$ so that

$$f(x) = 60x^3(1-x)^2 \quad \text{for } 0 \leq x \leq 1.$$

We could integrate $f(\cdot)$ to find $F(\cdot)$ and then try to use the inverse transform approach.

But no analytic expression for $F^{-1}(\cdot)$ so let's use the acceptance-rejection algorithm instead.

1. First choose $g(y)$: let's take $g(y) = 1$ for $y \in [0, 1]$, i.e., $Y \sim U(0, 1)$
2. now find $a$. Recall we must have

   $$\frac{f(x)}{g(x)} \leq a \text{ for all } x,$$

   which implies

   $$60x^3(1-x)^2 \leq a \text{ for all } x \in [0, 1].$$

   So take $a = 3$.

   Easy to check that this value works.

## Example: Generating a Beta$(a, b)$ Random Variable

We then have the following A-R algorithm.

```
generate $Y \sim U(0,1)$
generate $U \sim U(0,1)$
while $U > 20\,Y^3(1 - Y)^2$
    generate $Y$
    generate $U$
set $X = Y$
```

## Efficiency of the Acceptance-Rejection Algorithm

Let $N$ be the number of loops in the A-R algorithm until acceptance.

As before, let $B$ be the event $U \leq f(Y)/ag(Y)$

- saw earlier that $P(B) = 1/a$.

**Question:** What is the distribution of $N$?

**Question:** What is $\mathbb{E}[N]$?

**Question:** How should we choose $a$?

**Question:** How should we choose $g(\cdot)$?

## Other Methods for Generating Univariate Random Variables

Suppose we want to simulate a value of a random variable, $X$, and we know that

$$X \sim g(Y_1, \ldots, Y_n)$$

for some random variables $Y_1, \ldots, Y_n$ and some **function** $g(\cdot)$

- note that the $Y_i$'s need not necessarily be IID.

If we know how to generate $(Y_1, \ldots, Y_n)$ then can generate $X$ by:

1. Generating $(Y_1, \ldots, Y_n)$
2. Setting $X = g(Y_1, \ldots, Y_n)$.

## Generating Normal Random Variables

Typically rely on software packages to generate normal random variables.

Nonetheless worthwhile understanding how to do this.

First note that if $Z \sim \mathsf{N}(0,1)$ then

$$X := \mu + \sigma Z \sim \mathsf{N}(\mu, \sigma^2)$$

so only need to concern ourselves with generating $\mathsf{N}(0,1)$ random variables.

One possibility for doing this is to use the inverse transform method

- but would have to compute $F_z^{-1}(\cdot) := \Phi^{-1}(\cdot)$ numerically.

Other approaches for generating $\mathsf{N}(0,1)$ random variables include:

1. The Box-Muller method
2. The Polar method
3. Rational approximations.

Could also the A-R algorithm.

## The Box Muller Algorithm

The Box-Muller algorithm uses two IID $U(0,1)$ random variables to produce two IID $\mathsf{N}(0,1)$ random variables. It works as follows:

```
generate $U_1$ and $U_2$ IID $U(0,1)$
set
```

$$\boxed{\begin{aligned} X &= \sqrt{-2\log(U_1)}\ \cos(2\pi U_2) \\ Y &= \sqrt{-2\log(U_1)}\ \sin(2\pi U_2). \end{aligned}}$$

## Rational Approximations

Let $X \sim \mathsf{N}(0,1)$ and suppose $U \sim U(0,1)$. The inverse transform method then seeks $x_u = \Phi^{-1}(U)$.

Finding $\Phi^{-1}$ in closed form is not possible but instead, we can instead use rational approximations to $\Phi^{-1}$

- these are very accurate and efficient methods for estimating $x_u$.

**e.g.** For $0.5 \leq u \leq 1$

$$x_u \;\approx\; t - \frac{a_0 + a_1 t}{1 + b_1 t + b_2 t^2}$$

where $a_0, a_1, b_1$ and $b_2$ are constants, and $t = \sqrt{-2\log(1-u)}$.
The error is bounded in this case by $.003$.

Even more accurate approximations are available, and since they are very fast, many packages use them for generating normal random variables.

## The Multivariate Normal Distribution

If $\mathbf{X}$ multivariate normal with mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$ then write

$$\mathbf{X} \sim \mathsf{MN}_n(\boldsymbol{\mu}, \boldsymbol{\Sigma}).$$

Standard multivariate normal: $\boldsymbol{\mu} = \mathbf{0}$ and $\boldsymbol{\Sigma} = \mathbf{I_n}$, the $n \times n$ identity matrix.

PDF of $\mathbf{X}$ given by

$$f(\mathbf{x}) \;=\; \frac{1}{(2\pi)^{n/2}|\boldsymbol{\Sigma}|^{1/2}}\; e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top\,\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})} \tag{2}$$

where $|\cdot|$ denotes the determinant.

Characteristic function satisfies

$$\phi_{\mathbf{X}}(s) \;=\; \mathsf{E}\left[e^{is^\top\mathbf{X}}\right] \;=\; e^{is^\top\boldsymbol{\mu}\;-\;\frac{1}{2}s^\top\boldsymbol{\Sigma}s}.$$

# The Multivariate Normal Distribution

Let $\mathbf{X_1} = (X_1, \ldots, X_k)^\top$ and $\mathbf{X_2} = (X_{k+1}, \ldots, X_n)^\top$ be a partition of $\mathbf{X}$ with

$$\boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{pmatrix} \quad \text{and} \quad \boldsymbol{\Sigma} = \begin{pmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{pmatrix}.$$

Then marginal distribution of a multivariate normal random vector is itself (multivariate) normal. In particular, $\mathbf{X_i} \sim \mathsf{MN}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_{ii})$, for $i = 1, 2$.

Assuming $\boldsymbol{\Sigma}$ is positive definite, the conditional distribution of a multivariate normal distribution is also a (multivariate) normal distribution. In particular,

$$\mathbf{X_2} \mid \mathbf{X_1} = \mathbf{x_1} \sim \mathsf{MN}(\boldsymbol{\mu}_{\mathbf{2.1}}, \boldsymbol{\Sigma}_{2.1})$$

where

$$\boldsymbol{\mu}_{2.1} = \boldsymbol{\mu}_2 + \boldsymbol{\Sigma}_{21}\, \boldsymbol{\Sigma}_{11}^{-1}\, (\mathbf{x_1} - \boldsymbol{\mu}_1)$$

$$\boldsymbol{\Sigma}_{2.1} = \boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{\Sigma}_{12}.$$

## Generating MN Distributed Random Vectors

Suppose we wish to generate $\mathbf{X} = (X_1, \ldots, X_n)$ where $\mathbf{X} \sim \mathsf{MN}_n(\mathbf{0}, \mathbf{\Sigma})$

- it is then easy to handle the case where $\mathsf{E}[\mathbf{X}] \neq \mathbf{0}$.

Let $\mathbf{Z} = (Z_1, \ldots, Z_n)^\top$ where $Z_i \sim \mathsf{N}(0, 1)$ and IID for $i = 1, \ldots, n$.

If $\mathbf{C}$ an $(n \times m)$ matrix then

$$\mathbf{C}^\top \mathbf{Z} \sim \mathsf{MN}(0, \mathbf{C}^\top \mathbf{C}).$$

Problem therefore reduces to finding $\mathbf{C}$ such that $\mathbf{C}^\top \mathbf{C} = \mathbf{\Sigma}$.

Usually find such a matrix, $\mathbf{C}$, via the Cholesky decomposition of $\mathbf{\Sigma}$.

## The Cholesky Decomposition of a Symmetric PD Matrix

Any symmetric positive-definite matrix, $\mathbf{M}$, may be written as

$$\mathbf{M} = \mathbf{U}^\top \mathbf{D} \mathbf{U}$$

where:

- $\mathbf{U}$ is an upper triangular matrix
- $\mathbf{D}$ is a diagonal matrix with positive diagonal elements.

Since $\mathbf{\Sigma}$ is symmetric positive-definite, can therefore write

$$\begin{aligned} \mathbf{\Sigma} &= \mathbf{U}^\top \mathbf{D} \mathbf{U} \\ &= (\mathbf{U}^\top \sqrt{\mathbf{D}})(\sqrt{\mathbf{D}}\mathbf{U}) \\ &= (\sqrt{\mathbf{D}}\mathbf{U})^\top (\sqrt{\mathbf{D}}\mathbf{U}). \end{aligned}$$

$\mathbf{C} = \sqrt{\mathbf{D}}\mathbf{U}$ therefore satisfies $\mathbf{C}^\top \mathbf{C} = \mathbf{\Sigma}$

- **C** is called the Cholesky Decomposition of $\mathbf{\Sigma}$.

# The Cholesky Decomposition in Matlab

Easy to compute the Cholesky decomposition of a symmetric positive-definite matrix in Matlab using the chol command

- so also easy to simulate multivariate normal random vectors in Matlab.

**Sample Matlab Code**

```
>> Sigma = [1.0 0.5 0.5;
            0.5 2.0 0.3;
            0.5 0.3 1.5];

>> C = chol(Sigma);
>> Z = randn(3,1000000);
>> X = C’*Z;
>> cov(X’)

ans =
    0.9972    0.4969    0.4988
    0.4969    1.9999    0.2998
    0.4988    0.2998    1.4971
```

## The Cholesky Decomposition in `Matlab` and `R`

Must be very careful in `Matlab` and `R` to pre-multiply $\mathbf{Z}$ by $\mathbf{C}^\top$ and not $\mathbf{C}$.

Some languages take $\mathbf{C}^\top$ to be the Cholesky Decomposition rather $\mathbf{C}$

- must therefore always know what convention your programming language / package is using.

Must also be careful that $\mathbf{\Sigma}$ is indeed a genuine variance-covariance matrix.

## Simulating Poisson Processes

A Poisson process, $N(t)$, with intensity $\lambda$ is a process such that

$$P\left(N(t) = r\right) = \frac{(\lambda t)^r e^{-\lambda t}}{r!}.$$

For a Poisson process the **numbers of arrivals in non-overlapping intervals are independent** and the distribution of the number of arrivals in an interval only depends on the length of the interval.

The Poisson process is good for modeling many phenomena including the emission of particles from a radioactive source and the arrivals of customers to a queue.

The $i^{th}$ inter-arrival time, $X_i$, is defined to be the interval between the $(i-1)^{th}$ and $i^{th}$ arrivals of the Poisson process.

Easy to see the $X_i$'s are IID $\sim$ Exp$(\lambda)$

- so can simulate a Poisson process by simply generating the Exp$(\lambda)$ inter-arrival times, $X_i$.

Following algorithm simulates the first $T$ time units of a Poisson process:

## Simulating a Poisson Processes

**Simulating $T$ Time Units of a Poisson Process**

```
set $t = 0$, $I = 0$
generate $U$
set $t = t - \log(U)/\lambda$
while $t < T$
    set $I = I + 1$, $S(I) = t$
    generate $U$
    set $t = t - \log(U)/\lambda$
```

# The Non-Homogeneous Poisson Process

Obtain a non-homogeneous Poisson process, $N(t)$, by relaxing assumption that the intensity, $\lambda$, is constant.

If $\lambda(t) \geq 0$ is the intensity of the process at time $t$, then we say $N(t)$ is a non-homogeneous Poisson process with intensity $\lambda(t)$.

Define the function $m(t)$ by

$$m(t) := \int_0^t \lambda(s)\ ds.$$

Can be shown that $N(t+s) - N(t)$ is a Poisson random variable with parameter $m(t+s) - m(t)$, i.e.,

$$P\left(N(t+s) - N(t) = r\right) \ = \ \frac{\exp\left(-m_{t,s}\right)\left(m_{t,s}\right)^r}{r!}$$

where $m_{t,s} := m(t+s) - m(t)$.

## Simulating a Non-Homogeneous Poisson Process

Before we describe the thinning algorithm for simulating a non-homogeneous Poisson process, first need the following proposition.

**Proposition.** Let $N(t)$ be a Poisson process with constant intensity $\lambda$. Suppose that an arrival that occurs at time $t$ is counted with probability $p(t)$, independently of what has happened beforehand.

Then the process of counted arrivals is a non-homogeneous Poisson process with intensity $\lambda(t) = \lambda p(t)$. $\square$

Suppose now $N(t)$ is a non-homogeneous Poisson process with intensity $\lambda(t)$ and that there exists a $\lambda$ such that $\lambda(t) \leq \lambda$ for all $t \leq T$.

Then we can use the following algorithm, based on Proposition 1, to simulate $N(t)$.

# Simulating a Non-Homogeneous Poisson Process

## The Thinning Algorithm for Simulating $T$ Time Units of a NHPP

```
set $t = 0,\ I = 0$
generate $U_1$
set $t = t - \log(U_1)/\lambda$
while $t < T$

    generate $U_2$
    if $U_2 \leq \lambda(t)/\lambda$ then

        set $I = I + 1,\ S(I) = t$
    generate $U_1$
    set $t = t - \log(U_1)/\lambda$
```

## Brownian Motion

**Definition.** A stochastic process, $\{X_t : t \geq 0\}$, is a Brownian motion with parameters $(\mu, \sigma)$ if

1. For $0 < t_1 < t_2 < \ldots < t_{n-1} < t_n$

$$(X_{t_2} - X_{t_1}),\ (X_{t_3} - X_{t_2}), \ldots,\ \ (X_{t_n} - X_{t_{n-1}})$$

   are mutually independent.
2. For $s > 0$, $X_{t+s} - X_t \sim \mathsf{N}(\mu s, \sigma^2 s)$ and
3. $X_t$ is a continuous function of $t$ w.p. 1.

Say that $X$ is a $B(\mu, \sigma)$ Brownian motion with drift, $\mu$, and volatility, $\sigma$.

When $\mu = 0$ and $\sigma = 1$ we have a standard Brownian motion (SBM).

If $X \sim B(\mu, \sigma)$ and $X_0 = x$ then can write

$$X_t = x + \mu t + \sigma B_t.$$

## Simulating a Brownian Motion

**Simulating a Standard Brownian Motion at Times $t_1 < t_2 < \ldots < t_n$**

```
set $t_0 = 0$, $B_{t_0} = 0$
for $i = 1$ to $n$
    generate $X \sim \mathsf{N}(0, t_i - t_{i-1})$
    set $B_{t_i} = B_{t_{i-1}} + X$
```

**Question:** Can you suggest another method to generate $B_{t_i}$ for $t_1 < t_2 < \ldots < t_n$?

## Geometric Brownian Motion

**Definition.** A stochastic process, $\{X_t : t \geq 0\}$, is a $(\mu, \sigma)$ geometric Brownian motion (GBM) if

$$\log(X) \sim \mathsf{B}(\mu - \sigma^2/2,\ \sigma).$$

We write $X \sim \mathsf{GBM}(\mu, \sigma)$ and call $\mu$ the drift and $\sigma$ the volatility.

Note if $X \sim GBM(\mu, \sigma)$, then $X_t \sim \mathsf{LN}\left((\mu - \sigma^2/2)t,\ \sigma^2 t\right)$.

**Question:** How would you simulate $X_{t_i}$ for $t_1 < t_2 < \ldots < t_n$?

## Modelling Stock Prices as Geometric Brownian Motion

Suppose $X \sim \text{GBM}(\mu, \sigma)$. Then:

1. If $X_t > 0$, then $X_{t+s} > 0$ for any $s > 0$ so limited liability is not violated.
2. Distribution of $\frac{X_{t+s}}{X_t}$ only depends on $s$
   - so distribution of **returns** from one period to the next only depends on the length of the period.

This suggests that GBM might be a reasonable model for stock prices.

Will often model stock prices as GBM's and will use the following notation:

- $S_0$ is the known stock price at $t = 0$
- $S_t$ is the random stock price at time $t$ and satisfies

$$S_t = S_0 \, e^{(\mu - \sigma^2/2)t + \sigma B_t}.$$

so that

$$S_{t+\Delta t} = S_t \, e^{(\mu - \sigma^2/2)\Delta t + \sigma (B_{t+\Delta t} - B_t)}.$$

## E.G: Parameter Uncertainty and Hedging in Black-Scholes

Now consider the use of the Black-Scholes model to hedge a vanilla European call option in the model.

Will assume that assumptions of Black-Scholes are correct:

- Security price has GBM dynamics
- Possible to trade continuously at no cost
- Borrowing and lending at the risk-free rate are also possible.

Then possible to dynamically replicate payoff of the call option using a self-financing (s.f.) trading strategy

- initial value of this s.f. strategy is the famous Black-Scholes arbitrage-free price of the option.

The s.f. replication strategy requires **continuous** delta-hedging of the option but of course not practical to do this.

Instead we hedge **periodically** – this results in some **replication error**

- but this error goes to $0$ as the interval between rebalancing goes to $0$.

## E.G: Parameter Uncertainty and Hedging in Black-Scholes

$P_t$ denotes time $t$ value of the discrete-time s.f. strategy and $C_0$ denotes initial value of the option.

The replicating strategy is then satisfies

$$
\begin{aligned}
P_0 &:= C_0 && (3)\\
P_{t_{i+1}} &= P_{t_i} + \left(P_{t_i} - \delta_{t_i} S_{t_i}\right) r\Delta t + \delta_{t_i}\left(S_{t_{i+1}} - S_{t_i} + qS_{t_i}\Delta t\right) && (4)
\end{aligned}
$$

where:

- $\Delta t := t_{i+1} - t_i$
- $r =$ risk-free interest rate
- $q$ is the dividend yield
- $\delta_{t_i}$ is the Black-Scholes delta at time $t_i$
  - a function of $S_{t_i}$ and some assumed implied volatility, $\sigma_{imp}$.

Note that (3) and (4) respect the s.f. condition.

## E.G: Parameter Uncertainty and Hedging in Black-Scholes

Stock prices are simulated assuming $S_t \sim \mathsf{GBM}(\mu, \sigma)$ so that

$$S_{t+\Delta t} \;=\; S_t e^{(\mu - \sigma^2/2)\Delta t + \sigma\sqrt{\Delta t} Z}$$

where $Z \sim \mathsf{N}(0,1)$.

In the case of a short position in a call option with strike $K$ and maturity $T$, the final trading P&L is then defined as

$$\mathsf{P\&L} \;:=\; P_T - (S_T - K)^+ \tag{5}$$

where $P_T$ is the terminal value of the replicating strategy in (4).

In the Black-Scholes world we have $\sigma = \sigma_{imp}$ and the P&L $= 0$ along every price path in the limit as $\Delta t \to 0$.

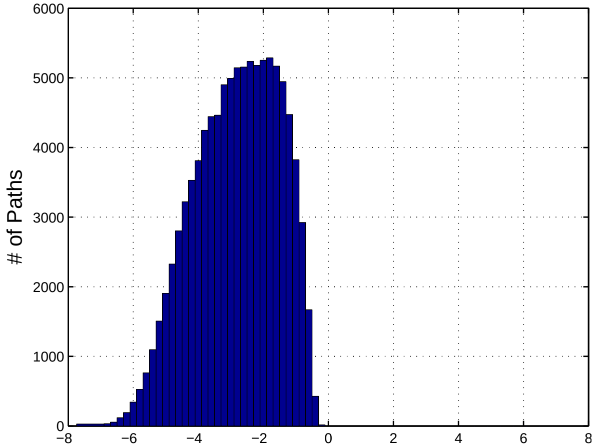
In practice, however, we cannot know $\sigma$ and so the market (and hence the option hedger) has no way to ensure a value of $\sigma_{imp}$ such that $\sigma = \sigma_{imp}$.

## E.G: Parameter Uncertainty and Hedging in Black-Scholes

This has interesting implications for the trading P$\&$L: it means we cannot exactly replicate the option even if all of the assumptions of Black-Scholes are correct!

In figures on next two slides we display histograms of the P&L in (5) that results from simulating 100k sample paths of the underlying price process with $S_0 = K = \$100$.

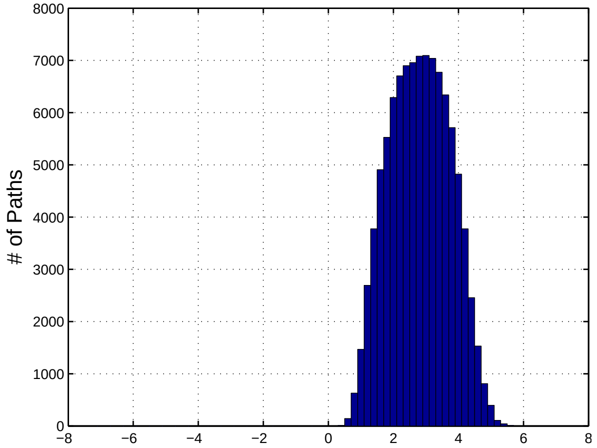
## E.G: Parameter Uncertainty and Hedging in Black-Scholes



Histogram of delta-hedging P&L with true vol. = 30% and implied vol. = 20%.

Option hedger makes substantial loses. **Why?**

## E.G: Parameter Uncertainty and Hedging in Black-Scholes



Histogram of delta-hedging P&L with true vol. = 30% and implied vol. = 40%.

Option hedger makes substantial gains. **Why?**

## E.G: Parameter Uncertainty and Hedging in Black-Scholes

Clearly then this is a situation where substantial errors in the form of non-zero hedging P&L's are made

- and this can only be due to the use of incorrect model parameters.

This example is intended to highlight the importance of not just having a good model but also having the correct model parameters.

The payoff from delta-hedging an option is in general path-dependent.

Can be shown that the payoff from continuously delta-hedging an option satisfies

$$\text{P\&L} = \int_0^T \frac{S_t^2}{2} \frac{\partial^2 V_t}{\partial S^2} \left( \sigma_{imp}^2 - \sigma_t^2 \right) \, dt$$

where $V_t$ is the time $t$ value of the option and $\sigma_t$ is the realized instantaneous volatility at time $t$.

We recognize the term $\frac{S_t^2}{2} \frac{\partial^2 V_t}{\partial S^2}$ as the dollar gamma

- always positive for a vanilla call or put option.

## E.G: Parameter Uncertainty and Hedging in Black-Scholes

Returning to s.f. trading strategy of (3) and (4), note that we can choose any model we like for the security price dynamics

- e.g. other diffusions or jump-diffusion models.

It is interesting to simulate these alternative models and to then observe what happens to the replication error from (3) and (4).

It is common to perform numerical experiments like this when using a model to price and hedge a particular security.

Goal then is to understand how robust the hedging strategy (based on the given model) is to alternative price dynamics that might prevail in practice.

Given the appropriate data, one can also back-test the performance of a model on realized historical price data to assess its hedging performance.