Generating Random Variables and Stochastic Processes

In these lecture notes we describe the principal methods that are used to generate random variables, taking as given a good $U(0,1)$ random variable generator. We begin with Monte-Carlo integration and then describe the main methods for random variable generation including inverse-transform, composition and acceptance-rejection. We also describe the generation of normal random variables and multivariate normal random vectors via the Cholesky decomposition. We end with a discussion of how to generate (non-homogeneous) Poisson processes as well (geometric) Brownian motions.

1 Monte Carlo Integration

Monte-Carlo simulation can also be used for estimating integrals and we begin with one-dimensional integrals. Suppose then that we want to compute

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

If we cannot compute θ analytically, then we could use numerical methods. However, we can also use simulation and this can be especially useful for high-dimensional integrals. The key observation is to note that $\theta = \mathbb{E}[g(U)]$ where $U \sim U(0,1)$. We can use this observation as follows:

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

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

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

Example 1 Suppose we wish to estimate $\int_0^1 x^3\ dx$ using simulation. We know the exact answer is $1/4$ but we can also estimate this using simulation. In particular if we generate n $U(0,1)$ independent variables, cube them and then take their average then we will have an unbiased estimate. ∎

Example 2 We wish to estimate $\theta = \int_1^3 (x^2 + x)\ dx$ again using simulation. Once again we know the exact answer (it's 12.67) but we can also estimate it by noting that

$$\theta \ = \ 2\int_1^3 \frac{x^2 + x}{2}\ dx \ = \ 2\mathbb{E}[X^2 + X]$$

where $X \sim U(1,3)$. So we can estimate θ by generating n IID $U(0,1)$ random variables, converting them (how?) to $U(1,3)$ variables, $X_1, \ldots, X_n$, and then taking $\widehat{\theta}_n := 2\sum_i (X_i^2 + X_i)/n$. ∎

1.1 Multi-Dimensional Monte Carlo Integration

Suppose now that we wish to approximate

$$\theta = \int_0^1 \int_0^1 g(x_1, x_2)\ dx_1 dx_2.$$

Then we can write $\theta = \mathbb{E}[g(U_1, U_2)]$ where U_1, U_2 are IID $U(0,1)$ random variables. Note that the joint PDF satisfies $f_{u_1,u_2}(u_1, u_2) = f_{u_1}(u_1) f_{u_2}(u_2) = 1$ on $[0,1]^2$. As before we can estimate θ using simulation by performing the following steps:

1. Generate n independent bivariate vectors (U_1^i, U_2^i) for $i = 1, \ldots, n$, with all U_j^i's IID $U(0,1)$.
2. Compute $g(U_1^i, U_2^i)$ for $i = 1, \ldots, n$
3. Estimate θ with

$$\widehat{\theta}_n \quad = \quad \frac{g(U_1^1, U_2^1) + \ \ldots\ + g(U_1^n, U_2^n)}{n}$$

As before, the SLLN justifies this approach and guarantees that $\widehat{\theta}_n \to \theta$ w.p. 1 as $n \to \infty$.

Example 3 (Computing a Multi-Dimensional Integral)
We can use Monte Carlo to estimate

$$\begin{aligned} \theta &:= \int_0^1 \int_0^1 (4x^2 y + y^2)\ dxdy \\ &= \mathbb{E}\left[4X^2 Y + Y^2\right] \end{aligned}$$

where X, Y are IID $U(0,1)$. (The true value of θ is easily calculated to be 1.) ∎

We can also apply Monte Carlo integration to more general problems. For example, if 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 we observe that $\theta = \mathbb{E}[g(X,Y)]$ where X, Y have joint density $f(x,y)$. To estimate θ using simulation we simply generate n random vectors (X,Y) with joint density $f(x,y)$ and then estimate θ with

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

---

2 Generating Univariate Random Variables

We will study a number of methods for generating univariate random variables. The three principal methods are the *inverse transform* method, the *composition* method and the *acceptance-rejection* method. All of these methods rely on having a (good) $U(0,1)$ random number generator available which we assume to be the case.

2.1 The Inverse Transform Method

The Inverse Transform Method for Discrete Random Variables

Suppose X is a discrete random variable with probability mass function (PMF)

$$X = \begin{cases} x_1, & \text{w.p.} \quad p_1 \\ x_2, & \text{w.p.} \quad p_2 \\ x_3, & \text{w.p.} \quad p_3 \end{cases}$$

where $p_1 + p_2 + p_3 = 1$. We would like to generate a value of X and we can do this by using our $U(0,1)$ generator as follows. First generate U and then set

$$X = \begin{cases} x_1, & \text{if} \quad 0 \le U \le p_1 \\ x_2, & \text{if} \quad p_1 < U \le p_1 + p_2 \\ x_3, & \text{if} \quad p_1 + p_2 < U \le 1. \end{cases}$$

We can easily check that this is correct: note that $\mathbf{P}(X = x_1) = \mathbf{P}(0 \le U \le p_1) = p_1$ since U is $U(0,1)$. The same is true for $\mathbf{P}(X = x_2)$ and $\mathbf{P}(X = x_3)$.

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

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

Then to generate a sample value of X we:

1. Generate U
2. Set $X = x_j$ if $\sum_{i=1}^{j-1} p_i < U \le \sum_{i=1}^{j} p_i$. That is, we set $X = x_j$ if $F(x_{j-1}) < U \le F(x_j)$.

If n is large, then we might want to search for x_j more efficiently, however!

Example 4 (Generating a Geometric Random Variable)
Suppose X is geometric with parameter p so that $\mathbf{P}(X = n) = (1-p)^{n-1}p$. Then we can generate X as follows:

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

   In particular, we set $X = \text{Int}\left(\frac{\log(U)}{\log(1-p)}\right) + 1$ where $\text{Int}(y)$ denotes the integer part of y.

You should convince yourself that this is correct! How does this compare to the coin-tossing method for generating X? ∎

Example 5 (Generating a Poisson Random Variable)
Suppose that X is Poisson(λ) so that $\mathbf{P}(X = n) = \exp(-\lambda)\ \lambda^n/n!$. We can generate X as follows:

1. Generate U
2. Set $X = j$ if $F(j-1) < U \le F(j)$.

How do we find j? We could use the following algorithm.

```
set j = 0, p = e^{-λ}, F = p
while U > F
        set p = λp/(j+1), F = F + p, j = j + 1
set X = j
```

Questions: How much work does this take? What if λ is large? Can we find j more efficiently?
Answer (to last question): Yes by checking if j is close to λ first.

Further questions: Why might this be useful? How much work does this take? ∎

The Inverse Transform Method for Continuous Random Variables

Suppose now that X is a continuous random variable and we want to generate a value of X. Recall that 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 $X = F_x^{-1}(U)$

We need to prove that this algorithm actually works! But this follows immediately since

$$\mathbf{P}(X \leq x) = \mathbf{P}(F_x^{-1}(U) \leq x) = \mathbf{P}(U \leq F_x(x)) = F_x(x)$$

as desired. 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 6 (Generating an Exponential Random Variable)
We wish to generate $X \sim$ Exp(λ). In this case $F_x(X) = 1 - e^{-\lambda x}$ so that $F_x^{-1}(u) = -\log(1-u)/\lambda$. We can generate X then by generating U and setting (why?) $X = -\log(U)/\lambda$. ∎

Example 7 (Generating a Gamma(n,λ) Random Variable)
We wish to generate $X \sim$ Gamma(n, λ) where n is a positive integer. Let X_i be IID $\sim$ exp(λ) for $i = 1, \ldots, n$. Note that if $Y := X_1 + \ldots + X_n$ then $Y \sim$ Gamma(n, λ). How can we use this observation to generate a sample value of Y? If n is not an integer, then we need another method to generate Y. ∎

Example 8 (Generating Order Statistics)
Order statistics are very important and have many applications in statistics, engineering and even finance. So 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:

Question: How do we generate a sample of $X_{(i)}$?

Method 1: Generate $U_1, \ldots, U_n$ and for each U_i compute $X_i = F_X^{-1}(U_i)$. We then order the X_i's and take the i^{th} smallest as our sample. How much work does this take?

Question: Can we do better?

Method 2: Sure, use the monotonicity of F!

Question: Can we do even better?

Method 3: Suppose $Z \sim$ 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 so that the density integrates to 1. How can we use this distribution?

Question: Can we do even better? ∎

Advantages of the Inverse Transform Method

There are two principal advantages to the inverse transform method:

1. Monotonicity: we 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. This property can be useful for some variance reduction techniques.

Disadvantages of the Inverse Transform Method

The principal disadvantage of the inverse transform method is that F_x^{-1} may not always be computable. For example, 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. For example, suppose $F_x(x) = x^5(1+x)^3/8$ for $0 \le x \le 1$. Then we cannot compute F_x^{-1}. One possible solution to these problems is to find F_x^{-1} numerically.

2.2 The Composition Approach

Another method for generating random variables is the composition approach. Suppose again that X has CDF F_x and that we wish to simulate a value of X. We can often write

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

where the F_j's are also CDFs, $p_j \ge 0$ 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. For example, suppose
$X \sim \mathsf{Hyperexponential}(\lambda_1, \alpha_1, \ldots, \lambda_n, \alpha_n)$ so that

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

where $\lambda_i,\ \alpha_i \ge 0$, and $\sum_i^n \alpha_i = 1$. Here $\alpha_i = 0$ for $i > n$. If it's difficult to simulate X directly using the inverse transform method then we could use the composition algorithm (see below) instead.

Composition Algorithm

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

We claim that X has the desired distribution!

Proof: We have

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

■

The proof actually suggests that the composition approach might arise naturally from 'sequential' type experiments. Consider the following example.

Example 9 (A Sequential Experiment)
Suppose we roll a dice and let $Y \in \{1, 2, 3, 4, 5, 6\}$ be the outcome. If If $Y = i$ then we generate Z_i from the distribution F_i and set $X = Z_i$.

What is the distribution of X? How do we simulate a value of X? ■

Example 10 (The Hyperexponential Distribution)
Let $X \sim$ Hyperexponential$(\lambda_1, \alpha_1, \lambda_2, \alpha_2)$ so that $f_x(x) = \alpha_1 \lambda_1 e^{-\lambda_1 x} + \alpha_2 \lambda_2 e^{-\lambda_2 x}$. In our earlier notation we have

$$\begin{aligned}
\alpha_1 &= p_1 \\
\alpha_2 &= p_2 \\
f_1(x) &= \lambda_1 e^{-\lambda_1 x} \\
f_2(x) &= \lambda_2 e^{-\lambda_2 x}
\end{aligned}$$

and the following algorithm will then generate a sample of X.

generate U_1
if $U_1 \leq p_1$ **then**
    set $i = 1$
else
    set $i = 2$
generate U_2
/* Now generate X from Exp(λ_i) */
set
$$X = -\frac{1}{\lambda_i} \log(U_2)$$

■

Question: How would you simulate a value of X if $F_x(x) = (x + x^3 + x^5)/3$?

When the decomposition

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

is not obvious, we can create an *artificial* decomposition by *splitting*.

Example 11 (Splitting)
Suppose

$$f_x(x) = \frac{1}{5}\ 1_{[-1,0]}(x) + \frac{6}{15}\ 1_{[0,2]}(x).$$

How do we simulate a value of X using vertical splitting? How would horizontal splitting work? ∎

2.3 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 either the inverse transform or composition algorithm. We might then wish to use the acceptance-rejection algorithm. Towards this end let Y be another random variable with density $g(\cdot)$ and suppose that it is easy to simulate a value of Y. If there exists a constant a such that

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

then we 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 $X = Y$

Question: Why must we have $a \geq 1$?

We must now prove that this algorithm does indeed work. We define B to be the event that Y has been accepted in the while loop, i.e., $U \leq f(Y)/ag(Y)$. We need to show that $\mathbf{P}(X \leq x) = F_x(x)$

Proof: First observe

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

We can compute $\mathbf{P}(B)$ as

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

while the numerator in (1) satisfies

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

Therefore $\mathbf{P}(X \leq x) = F_x(x)$, as required. ■

Example 12 (Generating a Beta(a, b) Random Variable)
Recall that X has a Beta(a, b) distribution if $f(x) = cx^{a-1}(1-x)^{b-1}$ for $0 \leq x \leq 1$. Suppose now that 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, for example, integrate $f(\cdot)$ to find $F(\cdot)$, and then try to use the inverse transform approach. However, it is hard to find $F^{-1}(\cdot)$. Instead, let's use the acceptance-rejection algorithm:

1. First choose $g(y)$: let's take $g(y) = 1$ for $y \in [0, 1]$, i.e., $Y \sim U(0, 1)$
2. Then find a. Recall that 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 $a = 3$. It is easy to check that this value works. We then have the following algorithm.

Algorithm

generate $Y \sim U(0, 1)$
generate $U \sim U(0, 1)$
while $U > 20Y^3(1-Y)^2$
    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, and as before, let B be the event that Y has been accepted in a loop, i.e. $U \leq f(Y)/ag(Y)$. We saw earlier that $\mathbf{P}(B) = 1/a$.

Questions:

1: What is the distribution of N?

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

How Do We Choose a?

$\mathbb{E}[N] = a$, so clearly we would like a to be as small as possible. Usually, this is just a matter of calculus.

Example 13 (Generating a Beta(a, b) Random Variable continued)
Recall the Beta$(4, 3)$ example with PDF $f(x) = 60x^3(1-x)^2$ for $x \in [0, 1]$. We chose $g(y) = 1$ for $y \in [0, 1]$ so that $Y \sim U(0, 1)$. The constant a had to satisfy

$$\frac{f(x)}{g(x)} \leq a \quad \text{for all } x \in [0, 1]$$

and we chose $a = 3$. We can do better by choosing

$$a = \max_{x \in [0,1]} \frac{f(x)}{g(x)} \approx 2.073.$$

■

How Do We Choose $g(\cdot)$?

We would like to choose $g(\cdot)$ to minimize the computational load. This can be achieved by taking $g(\cdot)$ 'close' to $f(\cdot)$. Then a will be close to 1 and so fewer iterations will be required in the A-R algorithm. There is a tradeoff, however: if $g(\cdot)$ is 'close' to $f(\cdot)$ then it will probably also be hard to simulate from $g(\cdot)$. So we often need to find a balance between having a 'nice' $g(\cdot)$ and a small value of a.

Acceptance-Rejection Algorithm for Discrete Random Variables

So far, we have expressed the A-R algorithm in terms of PDF's, thereby implicitly assuming that we are generating continuous random variables. However, the A-R algorithm also works for discrete random variables where we simply replace PDF's with PMF's. So suppose we wish to simulate a discrete random variable, X, with PMF, $p_i = \mathbf{P}(X = x_i)$. If we do not wish to use the discrete inverse transform method for example, then we can use the following version of the A-R algorithm. We assume that we can generate Y with PMF, $q_i = \mathbf{P}(Y = y_i)$, and that a satisfies $p_i/q_i \leq a$ for all i.

The Acceptance-Rejection Algorithm for Discrete Random Variables

generate Y with PMF q_i
generate U
while $U > \frac{p_Y}{aq_Y}$
    generate Y
    generate U
set $X = Y$

Generally, we would use this A-R algorithm when we can simulate Y efficiently.

Exercise 1 *(From* Simulation *by Sheldon M. Ross)*
Suppose $Y \sim$ Bin(n, p) and that we want to generate X where

$$\mathbf{P}(X = r) = \mathbf{P}(Y = r | Y \geq k)$$

for some fixed $k \leq n$. Assume $\alpha = \mathbf{P}(Y \geq k)$ has been computed.

1. *Give the inverse transform method for generating X.*
2. *Give another method for generating X.*
3. *For what values of α, small or large, would the algorithm in (2) be inefficient?*

Example 14 (Generating from a Uniform Distribution over a 2-D Region)
Suppose (X, Y) is uniformly distributed over a 2-dimensional area, A. How would you simulate a sample of (X, Y)? Note first that if $X \sim U(-1, 1)$, $Y \sim U(-1, 1)$ and X and Y are independent then (X, Y) is uniformly distributed over the region

$$A := \{(x, y) : -1 \leq x \leq 1, -1 \leq y \leq 1\}.$$

We can therefore (how?) simulate a sample of (X, Y) when A is a square. Suppose now that A is a circle of radius 1 centered at the origin. How do we simulate a sample of (X, Y) in that case? ∎

Remark 1 *The A-R algorithm is an important algorithm for generating random variables. Moreover it can be used to generate samples from distributions that are only known up to a constant. It is very inefficient in high-dimensions, however, which is why* Markov Chain Monte Carlo *(MCMC) algorithms are required.*

3 Other Methods for Generating Univariate Random Variables

Besides the inverse transform, composition and acceptance-rejection algorithms, there are a number of other important methods for generating random variables. We begin with the convolution method.

3.1 The Convolution Method

Suppose $X \sim Y_1 + Y_2 + \ldots + Y_n$ where the Y_i's are IID with CDF $F_y(\cdot)$. Suppose also that it's easy to generate the Y_i's. Then it is straightforward to generate a value of X:

1. Generate $Y_1, \ldots, Y_n$ that have CDF F_y
2. Set $X = Y_1 + \ldots + Y_n$

We briefly mentioned this earlier in Example 7 when we described how to generate a Gamma(λ, n) random variable. The convolution method is not always the most efficient method. Why?

More generally, 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_i 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 we can generate X by generating $(Y_1, \ldots, Y_n)$ and setting $X = g(Y_1, \ldots, Y_n)$. We saw such an application in Example 7.

Example 15 (Generating Lognormal Random Variables)
Suppose $X \sim \mathsf{N}(\mu, \sigma^2)$. Then $Y := \exp(X)$ has a lognormal distribution, i.e., $Y \sim \mathsf{LN}(\mu, \sigma^2)$. (Note $\mathbb{E}[Y] \neq \mu$ and $\mathrm{Var}(Y) \neq \sigma^2$.) How do we generate a log-normal random variable? ∎

Example 16 (Generating χ^2 Random Variables)
Suppose $X \sim \mathsf{N}(0, 1)$. Then $Y := X^2$ has a a chi-square distribution with 1 degree of freedom, i.e., $Y \sim \chi^2_1$.

Question: How would you generate a χ^2_1 random variable?

Suppose now that $X_i \sim \chi^2_1$ for $i = 1, \ldots, n$. Then $Y := X_1 + \ldots + X_n$ has a chi-square distribution with n degrees-of-freedom, i.e., $Y \sim \chi^2_n$.

Question: How would you generate a χ^2_n random variable? ∎

Example 17 (Generating t_n Random Variables)
Suppose $X \sim \mathsf{N}(0, 1)$ and $Y \sim \chi^2_n$ with X and Y independent. Then

$$Z := \frac{X}{\sqrt{\frac{Y}{n}}}$$

has a a t distribution with n degrees of freedom , i.e., $Z \sim t_n$.

Question: How would you generate a t_n random variable? ∎

Example 18 (Generating $F_{m,n}$ Random Variables)
Suppose $X \sim \chi^2_m$ and $Y \sim \chi^2_n$ with X and Y independent. Then

$$Z := \frac{\left(\frac{X}{m}\right)}{\left(\frac{Y}{n}\right)}$$

has an F distribution with m and n degrees of freedom, i.e., $Z \sim F_{m,n}$.

Question: How would you generate a $F_{m,n}$ random variable? ∎

---

4 Generating Normal Random Variables

While we typically rely on software packages to generate normal random variables for us, it is nonetheless worthwhile having an understanding of how to do this. We first note that if $Z \sim \mathsf{N}(0,1)$ then

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

so that we need only concern ourselves with generating $\mathsf{N}(0,1)$ random variables. One possibility for doing this is to use the inverse transform method. But we would then have to use numerical methods since we cannot find $F_z^{-1}(\cdot) := \Phi^{-1}(\cdot)$ in closed form. Other approaches for generating $\mathsf{N}(0,1)$ random variables include:

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

There are many other methods such as the A-R algorithm that could also be used to generate $N(0,1)$ random variables.

4.1 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:

The Box-Muller Algorithm for Generating Two IID $\mathbf{N}(0,1)$ Random Variables

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

$$X = \sqrt{-2\log(U_1)}\ \cos(2\pi U_2) \quad \text{and} \quad Y = \sqrt{-2\log(U_1)}\ \sin(2\pi U_2)$$

We now show that this algorithm does indeed produce two IID $\mathsf{N}(0,1)$ random variables, X and Y.

Proof: We need to show that

$$f(x,y) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right)\ \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{y^2}{2}\right)$$

First, make a change of variables:

$$\begin{aligned} R &:= \sqrt{X^2 + Y^2} \\ \theta &:= \tan^{-1}\left(\frac{Y}{X}\right) \end{aligned}$$

so R and θ are polar coordinates of (X, Y). To transform back, note $X = R\cos(\theta)$ and $Y = R\sin(\theta)$. Note also that $R = \sqrt{-2\log(U_1)}$ and $\theta = 2\pi U_2$. Since U_1 and U_2 are IID, R and θ are independent. Clearly $\theta \sim U(0, 2\pi)$ so $f_\theta(\theta) = 1/2\pi$ for $0 \le \theta \le 2\pi$. It is also easy to see that $f_R(r) = re^{-r^2/2}$ for $r \ge 0$, so that

$$f_{R,\theta}(r, \theta) = \frac{1}{2\pi} re^{-r^2/2}, \quad 0 \le \theta \le 2\pi, \ r \ge 0.$$

This implies

$$\begin{aligned} \mathbf{P}(X \le x_1, Y \le y_1) &= \mathbf{P}(R\cos(\theta) \le x_1, R\sin(\theta) \le y_1) \\ &= \int\int_A \frac{1}{2\pi} re^{-r^2/2} \, dr \, d\theta \end{aligned} \tag{2}$$

where $A = \{(r, \theta) : r\cos(\theta) \le x, \ r\sin(\theta) \le y\}$. We now transform back to (x, y) coordinates with

$$x = r\cos(\theta) \quad \text{and} \quad y = r\sin(\theta)$$

and note that $dx\, dy = r dr\, d\theta$, i.e., the Jacobian of the transformation is r. We then use (2) to obtain

$$\begin{aligned} \mathbf{P}(X \le x_1, Y \le y_1) &= \frac{1}{2\pi} \int_{-\infty}^{x_1} \int_{-\infty}^{y_1} exp\left(-\frac{(x^2 + y^2)}{2}\right) \, dxdy \\ &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x_1} \exp(-x^2/2) \, dx \ \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{y_1} \exp(-y^2/2) \, dy \end{aligned}$$

as required. ■

4.2 The Polar Method

One disadvantage of the Box-Muller method is that computing sines and cosines is inefficient. We can get around this problem using the polar method which is described in the algorithm below.

The Polar Algorithm for Generating Two IID $\mathbf{N}(0, 1)$ Random Variables

set $S = 2$
while $S > 1$
    generate U_1 and U_2 IID $U(0, 1)$
    set $V_1 = 2U_1 - 1$, $V_2 = 2U_2 - 1$ and $S = V_1^2 + V_2^2$
set

$$X = \sqrt{\frac{-2\log(S)}{S}} V_1 \quad \text{and} \quad Y = \sqrt{\frac{-2\log(S)}{S}} V_2$$

Can you see why this algorithm[1] works?

4.3 Rational Approximations

Let $X \sim \mathsf{N}(0, 1)$ and recall that $\Phi(x) = \mathbf{P}(X \le x)$ is the CDF of X. If $U \sim U(0, 1)$, then the inverse transform method seeks $x_u = \Phi^{-1}(U)$. Finding Φ^{-1} in closed form is not possible but instead, we can use *rational approximations*. These are very accurate and efficient methods for estimating x_u.

[1] See *Simulation* by Sheldon M. Ross for further details.

Example 19 (Rational Approximations)
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 (including Matlab) use them for generating normal random variables. ∎

---

5 The Multivariate Normal Distribution

If the n-dimensional vector $\mathbf{X}$ is multivariate normal with mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$ then we write

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

The standard multivariate normal has $\boldsymbol{\mu} = \mathbf{0}$ and $\boldsymbol{\Sigma} = \mathbf{I_n}$, the $n \times n$ identity matrix. The PDF of $\mathbf{X}$ is given by

$$f(\mathbf{x}) \;=\; \frac{1}{(2\pi)^{n/2}|\boldsymbol{\Sigma}|^{1/2}}\; e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top\, \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})} \tag{3}$$

where $|\cdot|$ denotes the determinant, and its characteristic function satisfies

$$\phi_{\mathbf{X}}(\mathbf{s}) \;=\; \mathrm{E}\left[e^{i\mathbf{s}^\top \mathbf{X}}\right] \;=\; e^{i\mathbf{s}^\top \boldsymbol{\mu} - \frac{1}{2}\mathbf{s}^\top \boldsymbol{\Sigma} s}. \tag{4}$$

Recall again our partition of $\mathbf{X}$ into $\mathbf{X_1} = (X_1, \ldots, X_k)^\top$ and $\mathbf{X_2} = (X_{k+1}, \ldots, X_n)^\top$. If we extend this notation naturally so that

$$\boldsymbol{\mu} \;=\; \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix} \qquad \text{and} \qquad \boldsymbol{\Sigma} \;=\; \begin{pmatrix} \boldsymbol{\Sigma_{11}} & \boldsymbol{\Sigma_{12}} \\ \boldsymbol{\Sigma_{21}} & \boldsymbol{\Sigma_{22}} \end{pmatrix}.$$

then we obtain the following results regarding the marginal and conditional distributions of $\mathbf{X}$.

Marginal Distribution

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

Conditional Distribution

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

$$\mathbf{X_2} \mid \mathbf{X_1} = \mathbf{x_1} \;\sim\; \mathsf{MN}(\mu_{\mathbf{2.1}}, \boldsymbol{\Sigma_{2.1}})$$

where $\mu_{2.1} = \mu_\mathbf{2} + \boldsymbol{\Sigma_{21}}\, \boldsymbol{\Sigma_{11}^{-1}}\, (\mathbf{x_1} - \mu_1)$ and $\boldsymbol{\Sigma_{2.1}} = \boldsymbol{\Sigma_{22}} - \boldsymbol{\Sigma_{21}}\boldsymbol{\Sigma_{11}^{-1}}\boldsymbol{\Sigma_{12}}$.

Linear Combinations

Linear combinations of multivariate normal random vectors remain normally distributed with mean vector and covariance matrix given by

$$\begin{aligned} \mathrm{E}\left[\mathbf{AX} + \mathbf{a}\right] &= \mathbf{A}\mathrm{E}\left[\mathbf{X}\right] \;+\; \mathbf{a} \\ \mathrm{Cov}(\mathbf{AX} + \mathbf{a}) &= \mathbf{A}\,\mathrm{Cov}(\mathbf{X})\,\mathbf{A}^\top. \end{aligned}$$

Estimation of Multivariate Normal Distributions

The simplest and most common method of estimating a multivariate normal distribution is to take the sample mean vector and sample covariance matrix as our estimators of $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$, respectively. It is easy to justify this choice since they are the *maximum likelihood* estimators. It is also common to take $n/(n-1)$ times the sample covariance matrix as an estimator of $\boldsymbol{\Sigma}$ as this estimator is known to be *unbiased*.

Testing Normality and Multivariate Normality

There are many tests that can be employed for testing normality of random variables and vectors. These include standard univariate tests and tests based on *QQplots*, as well *omnibus moment tests* based on whether the skewness and kurtosis of the data are consistent with a multivariate normal distribution. Section $3.1.4$ of *MFE* should be consulted for details on these tests.

5.1 Generating Multivariate Normally Distributed Random Vectors

Suppose that we wish to generate $\mathbf{X} = (X_1, \ldots, X_n)$ where $\mathbf{X} \sim \mathsf{MN}_n(\mathbf{0}, \boldsymbol{\Sigma})$. Note that it is then easy to handle the case where $\mathbb{E}[\mathbf{X}] \neq \mathbf{0}$. Let $\mathbf{Z} = (Z_1, \ldots, Z_n)^\top$ where the Z_i's are IID $\mathsf{N}(0,1)$ for $i = 1, \ldots, n$. If $\mathbf{C}$ is an $(n \times m)$ matrix then it follows that

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

Our problem therefore reduces to finding $\mathbf{C}$ such that $\mathbf{C}^\top \mathbf{C} = \boldsymbol{\Sigma}$. We can use the *Cholesky decomposition* of $\boldsymbol{\Sigma}$ to find such a matrix, $\mathbf{C}$.

The Cholesky Decomposition of a Symmetric Positive-Definite Matrix

A well known fact from linear algebra is that 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 and $\mathbf{D}$ is a diagonal matrix with positive diagonal elements. Since $\boldsymbol{\Sigma}$ is symmetric positive-definite, we can therefore write

$$\boldsymbol{\Sigma} \;=\; \mathbf{U}^\top \mathbf{D} \mathbf{U} \;=\; (\mathbf{U}^\top \sqrt{\mathbf{D}})(\sqrt{\mathbf{D}}\mathbf{U}) \;=\; (\sqrt{\mathbf{D}}\mathbf{U})^\top (\sqrt{\mathbf{D}}\mathbf{U}).$$

The matrix $\mathbf{C} = \sqrt{\mathbf{D}}\mathbf{U}$ therefore satisfies $\mathbf{C}^\top \mathbf{C} = \boldsymbol{\Sigma}$. It is called the Cholesky Decomposition of $\boldsymbol{\Sigma}$.

The Cholesky Decomposition in `Matlab` and `R`

It is easy to compute the Cholesky decomposition of a symmetric positive-definite matrix in `Matlab` and `R` using the *chol* command and so it is also easy to simulate multivariate normal random vectors. As before, let $\boldsymbol{\Sigma}$ be an $(n \times n)$ variance-covariance matrix and let $\mathbf{C}$ be its Cholesky decomposition. If $\mathbf{X} \sim \mathsf{MN}(\mathbf{0}, \boldsymbol{\Sigma})$ then we can generate random samples of $\mathbf{X}$ in `Matlab` as follows:

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
```

We must be very careful in Matlab[2] and R to pre-multiply Z by $\mathbf{C}^\top$ and not $\mathbf{C}$. We have the following algorithm for generating multivariate random vectors, $\mathbf{X}$.

Generating Correlated Normal Random Variables

generate $\mathbf{Z} \sim \text{MN}(\mathbf{0}, \mathbf{I})$
/* Now compute the Cholesky Decomposition */
compute $\mathbf{C}$ such that $\mathbf{C}^\top\mathbf{C} = \mathbf{\Sigma}$
set $\mathbf{X} = \mathbf{C}^\top\mathbf{Z}$

---

6 Simulating Poisson Processes

Recall that a Poisson process, $N(t)$, with intensity λ is a process such that

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

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, and it is easy to see that the X_i's are IID $\sim \text{Exp}(\lambda)$. In particular, this means we can simulate a Poisson process with intensity λ by simply generating the inter-arrival times, X_i, where $X_i \sim \text{Exp}(\lambda)$. We have the following algorithm for simulating the first T time units of a Poisson process:

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 $t = t - \log(U)/\lambda$

6.1 The Non-Homogeneous Poisson Process

A non-homogeneous Poisson process, $N(t)$, is obtained by relaxing the assumption that the intensity, λ, is constant. Instead we take it to be a deterministic function of time, $\lambda(t)$. More formally, if $\lambda(t) \geq 0$ is the intensity of the process at time t, then we say that $N(t)$ is a non-homogeneous Poisson process with intensity $\lambda(t)$. Define the function $m(t)$ by

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

[2]Unfortunately, some languages take $\mathbf{C}^\top$ to be the Cholesky Decomposition rather $\mathbf{C}$. You must therefore always be aware of exactly what convention your programming language / package is using.

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

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

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, we first need the following[3] proposition.

Proposition 1 *Let $N(t)$ be a Poisson process with constant intensity λ. 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 $\lambda(t) = \lambda p(t)$.*

Suppose now $N(t)$ is a non-homogeneous Poisson process with intensity $\lambda(t)$ and that there exists a λ such that $\lambda(t) \leq \lambda$ for all $t \leq T$. Then we can use the following algorithm, based on Proposition 1, to simulate $N(t)$.

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 $t = t - \log(U_1)/\lambda$

Questions

1. Can you give a more efficient version of the algorithm when there exists $\lambda > 0$ such that $\min_{0 \leq t \leq T} \lambda(t) \geq \lambda$?

2. Can you think of another algorithm for simulating a non-homogeneous Poisson process that is not based on thinning?

6.2 Credit Derivatives Models

Many credit derivatives models use *Cox* processes to model company defaults. A Cox process, $C(t)$, is similar to a non-homogeneous Poisson process except that the intensity function, $\lambda(t)$, is itself a stochastic process. However, conditional upon knowing $\lambda(t)$ for all $t \in [0, T]$, $C(t)$ is a non-homogeneous Poisson process. In credit derivatives models, bankruptcy of a company is often modelled as occurring on the first arrival in the Cox process where the intensity at time t, $\lambda(t)$, generally depends on the level of other variables in the economy. Such variables might include, for example, interest rates, credit ratings and stock prices, all of which are themselves random. An understanding of and ability to simulate non-homogeneous Poisson processes is clearly necessary for analyzing such credit derivatives models.

[3] A proof may be found in *Simulation* by Sheldon M. Ross.

7 Simulating (Geometric) Brownian Motion

Definition 1 *A stochastic process, $\{X_t : t \geq 0\}$, is a Brownian motion with parameters (μ, σ) 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 $s > 0$, $X_{t+s} - X_t \sim N(\mu s, \sigma^2 s)$ and*

3. *X_t is a continuous function of t w.p. 1.*

We say that X is a $B(\mu, \sigma)$ Brownian motion with drift, μ, and volatility, σ. When $\mu = 0$ and $\sigma = 1$ we have a **standard** Brownian motion (SBM). We will use B_t to denote a SBM and we will *always* assume (unless otherwise stated) that $B_0 = 0$. Note that if $X \sim B(\mu, \sigma)$ and $X_0 = x$ then we can write

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

where B is a SBM. We will usually write a $B(\mu, \sigma)$ Brownian motion in this way.

Remark 2 *Bachelier (1900) and Einstein (1905) were the first to explore Brownian motion from a mathematical viewpoint whereas Wiener (1920's) was the first to show that it actually exists as a well-defined mathematical entity.*

Questions

1. What is $\mathbb{E}[B_{t+s}B_s]$?

2. What is $\mathbb{E}[X_{t+s}X_s]$ where $X \sim B(\mu, \sigma)$?

3. Let B be a SBM and let $Z_t := |B_t|$. What is the CDF of Z_t for t fixed?

7.1 Simulating a Standard Brownian Motion

It is not possible to simulate an entire sample path of Brownian motion between 0 and T as this would require an infinite number of random variables. This is not always a problem, however, since we often only wish to simulate the value of Brownian motion at certain fixed points in time. For example, we may wish to simulate B_{t_i} for $t_1 < t_2 < \ldots < t_n$, as opposed to simulating B_t for every $t \in [0, T]$.

Sometimes, however, the quantity of interest, θ, that we are trying to estimate does indeed depend on the entire sample path of B_t in $[0, T]$. In this case, we can still estimate θ by again simulating B_{t_i} for $t_1 < t_2 < \ldots < t_n$ but where we now choose n to be very large. We might, for example, choose n so that $|t_{i+1} - t_i| < \epsilon$ for all i, where $\epsilon > 0$ is very small. By choosing ϵ to be sufficiently small, we hope to minimize the *numerical error* (as opposed to the *statistical error*), in estimating θ. We will return to this issue later in the course when we learn how to simulate stochastic differential equations (SDE's).

In either case, we need to be able to simulate B_{t_i} for $t_1 < t_2 < \ldots < t_n$ and for a fixed n. We will now see how to do this. The first observation we make is that

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

are mutually independent, and for $s > 0$, $B_{t+s} - B_t \sim \mathsf{N}(0, s)$. The idea then is as follows: we begin with $t_0 = 0$ and $B_{t_0} = 0$. We then generate B_{t_1} which we can do since $B_{t_1} \sim \mathsf{N}(0, t_1)$. We now generate B_{t_2} by first observing that $B_{t_2} = B_{t_1} + (B_{t_2} - B_{t_1})$. Then since $(B_{t_2} - B_{t_1})$ is independent of B_{t_1}, we can generate B_{t_2} by generating an $\mathsf{N}(0, t_2 - t_1)$ random variable and simply adding it to B_{t_1}. More generally, if we have already generated B_{t_i} then we can generate $B_{t_{i+1}}$ by generating an $\mathsf{N}(0, t_{i+1} - t_i)$ random variable and adding it to B_{t_i}. We have the following algorithm.

Simulating a Standard Brownian Motion

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 $B_{t_i} = B_{t_{i-1}} + X$

Remark 3 *It is very important that when you generate $B_{t_{i+1}}$, you do so* conditional *on the value of B_{t_i}. If you generate B_{t_i} and $B_{t_{i+1}}$ independently of one another then you are effectively simulating from* different *sample paths of the Brownian motion. This is not correct! In fact when we generate $(B_{t_1}, B_{t_2}, \ldots, B_{t_n})$ we are actually generating a* random vector *that does* not *consist of IID random variables.*

Simulating a $\mathbf{B}(\mu, \sigma)$ Brownian Motion

Suppose now that we want to simulate a $B(\mu, \sigma)$ BM, X, at the times $t_1, t_2, \ldots, t_{n-1}, t_n$. Then all we have to do is simulate an SBM, $(B_{t_1}, B_{t_2}, \ldots, B_{t_n})$, and use our earlier observation that $X_t = x + \mu t + \sigma B_t$.

Brownian Motion as a Model for Stock Prices?

There are a number of reasons why Brownian motion is **not** a good model for stock prices. They include

1. The limited liability of shareholders
2. The fact that people care about *returns*, not absolute prices so the IID increments property of BM should *not* hold for stock prices.

As a result, geometric Brownian Motion (GBM) is a much better model for stock prices.

7.2 Geometric Brownian Motion

Definition 2 *A stochastic process, $\{X_t : t \geq 0\}$, is a (μ, σ) geometric Brownian motion (GBM) if $\log(X) \sim B(\mu - \sigma^2/2,\ \sigma)$. We write $X \sim GBM(\mu, \sigma)$.*

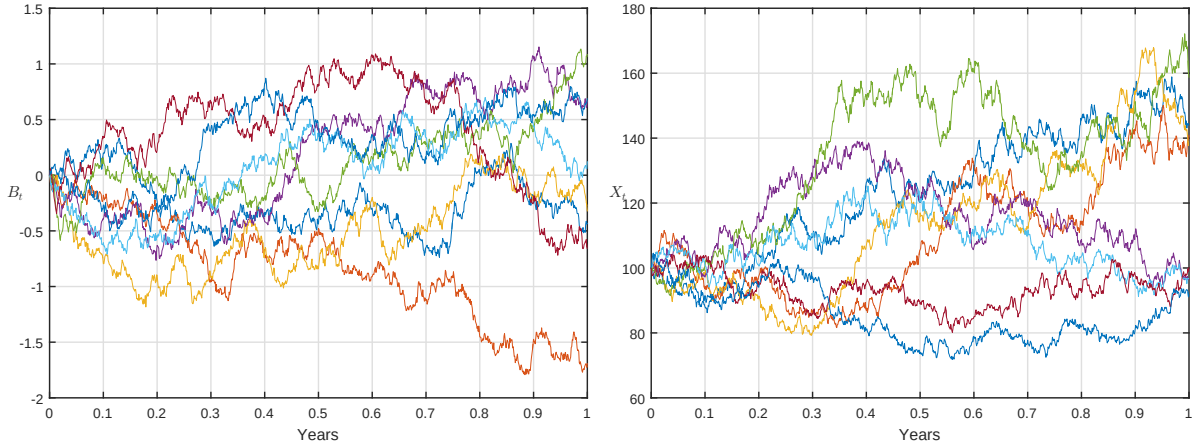
The following properties of GBM follow immediately from the definition of BM:

1. Fix $t_1, t_2, \ldots, t_n$. Then $\frac{X_{t_2}}{X_{t_1}},\ \frac{X_{t_3}}{X_{t_3}},\ \frac{X_{t_n}}{X_{t_{n-1}}}$ are mutually independent.
2. For $s > 0$, $\log\left(\frac{X_{t+s}}{X_t}\right) \sim \mathsf{N}\left((\mu - \sigma^2/2)s,\ \sigma^2 s\right)$.
3. X_t is continuous w.p. 1.

Again, we call μ the drift and σ the volatility. If $X \sim GBM(\mu, \sigma)$, then note that X_t has a lognormal distribution. In particular, if $X \sim \mathsf{GBM}(\mu, \sigma)$, then $X_t \sim \mathsf{LN}\left((\mu - \sigma^2/2)t,\ \sigma^2 t\right)$. In Figure 1 we have plotted some sample paths of Brownian and geometric Brownian motions.

Question: How would you simulate a sample path of $GBM(\mu, \sigma^2)$ at the fixed times $0 < t_1 < t_2 < \ldots < t_n$?

Answer: Simulate $\log(X_{t_i})$ first and then take exponentials! (See below for more details.)



(a) Brownian motion

(b) Geometric Brownian motion

Figure 1: Sample paths of Brownian motion, B_t, and geometric Brownian motion (GBM), $X_t = X_0\, e^{(\mu-\sigma^2/2)t+\sigma B_t}$. Parameters for the GBM were $X_0 = 100$, $\mu = 10\%$ and $\sigma = 30\%$.

Modelling Stock Prices as Geometric Brownian Motion

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

1. If $X_t > 0$, then X_{t+s} is always positive for any $s > 0$ so limited liability is not violated.
2. The distribution of $\frac{X_{t+s}}{X_t}$ only depends on s so the 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. In fact, we will often model stock prices as GBM's in this course, and we will generally 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

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

  where B is a standard BM. The drift is μ, σ is the volatility and S is a therefore a $GBM(\mu, \sigma)$ process that begins at S_0.

Questions

1. What is $\mathbb{E}[S_t]$?

2. What is $\mathbb{E}[S_t^2]$?

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

Suppose now that we wish to simulate $S \sim GBM(\mu, \sigma)$. Then we know

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

so that we can simulate $S_{t+\Delta t}$ conditional on S_t for any $\Delta t > 0$ by simply simulating an $\text{N}(0, \Delta t)$ random variable.

Example 20 (Simulating Delta-Hedging in a Black-Scholes Economy)
In this extended example we consider the use of the Black-Scholes model to hedge a vanilla European call option. Moreover, we will assume that the assumptions of Black-Scholes are correct so that the security price has GBM dynamics, it is possible to trade continuously at no cost and borrowing and lending at the risk-free rate are also possible. It is then possible to dynamically *replicate* the payoff of the call option using a self-financing (s.f.) trading strategy. The initial value of this s.f. strategy is the famous Black-Scholes arbitrage-free price of the option. The s.f. replication strategy requires the continuous *delta-hedging* of the option but of course it is not practical to do this and so instead we hedge periodically. (Periodic or discrete hedging then results in some *replication error* but this error goes to 0 as the time interval between re-balancing goes to 0.)

Towards this end, let P_t denote the time t value of the discrete-time s.f. strategy that attempts to replicate the option payoff and let C_0 denote the initial value of the option. The replicating strategy is then given by

$$\begin{aligned} P_0 &:= C_0 && (5)\\ P_{t_{i+1}} &= P_{t_i} + (P_{t_i} - \delta_{t_i} S_{t_i})\, r\Delta t + \delta_{t_i}\left(S_{t_{i+1}} - S_{t_i} + qS_{t_i}\Delta t\right) && (6)\end{aligned}$$

where $\Delta t := t_{i+1} - t_i$ is the length of time between re-balancing (assumed constant for all i), r is the annual risk-free interest rate (assuming per-period compounding), q is the dividend yield and δ_{t_i} is the Black-Scholes delta at time t_i. This delta is a function of S_{t_i} and some assumed implied volatility, σ_{imp} say. Note that (5) and (6) respect the self-financing condition. Stock prices are simulated assuming $S_t \sim \text{GBM}(\mu, \sigma)$ so that

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

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

$$\text{P\&L} := P_T - (S_T - K)^+ \quad (7)$$

where P_T is the terminal value of the replicating strategy in (6). In the Black-Scholes world we have $\sigma = \sigma_{imp}$ and the P&L will be 0 along every price path in the limit as $\Delta t \to 0$.

In practice, however, we do not know σ and so the market (and hence the option hedger) has no way to ensure a value of σ_{imp} such that $\sigma = \sigma_{imp}$. This has interesting implications for the trading P&L and it means in particular that we cannot exactly replicate the option even if all of the assumptions of Black-Scholes are correct. In Figure 2 we display histograms of the P&L in (7) that results from simulating 100,000 sample paths of the underlying price process with $S_0 = K = \$100$. (Other parameters and details are given below the figure.) In the case of the first histogram the true volatility was $\sigma = 30\%$ with $\sigma_{imp} = 20\%$ and the option hedger makes (why?) substantial loses. In the case of the second histogram the true volatility was $\sigma = 30\%$ with $\sigma_{imp} = 40\%$ and the option hedger makes (why?) substantial gains.

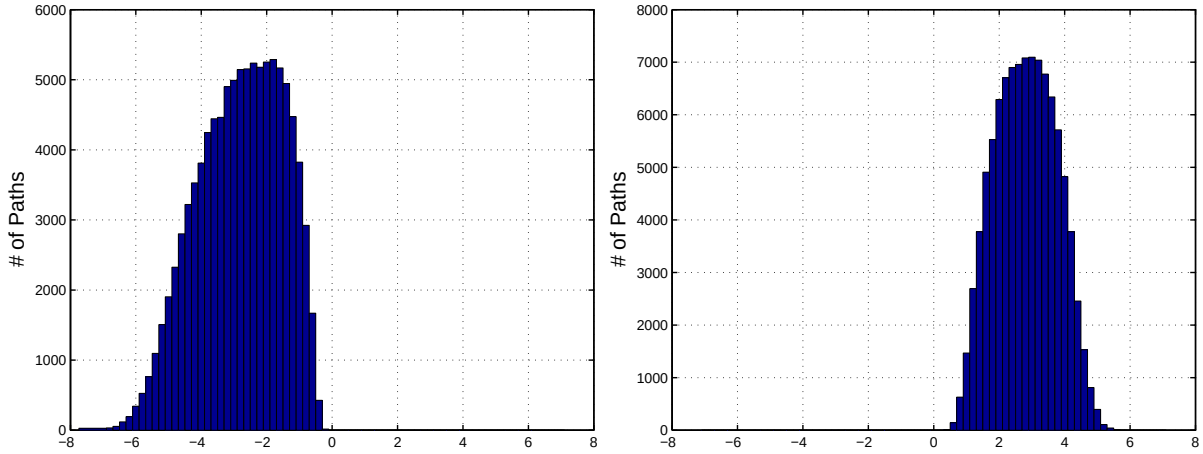
Clearly then this is a situation where substantial errors in the form of non-zero hedging P&L's are made and this can only be due to the use of incorrect model parameters. This example is intended[4] to highlight the importance of not just having a good model but also having the correct model parameters.

Note that the payoff from delta-hedging an option is in general *path-dependent*, i.e. it depends on the price path taken by the stock over the entire time interval. In fact, it can be shown that the payoff from continuously delta-hedging an option satisfies

$$\text{P\&L} = \int_0^T \frac{S_t^2}{2}\frac{\partial^2 V_t}{\partial S^2}\left(\sigma_{imp}^2 - \sigma_t^2\right) dt \quad (8)$$

where V_t is the time t value of the option and σ_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*. It is always positive for a call or put option, but it goes to zero as the

[4] We do acknowledge that this example is somewhat contrived in that if the true price dynamics really were GBM dynamics then we could estimate σ_{imp} perfectly and therefore exactly replicate the option payoff (in the limit of continuous trading). That said, it can also be argued that this example is not at all contrived: options traders in practice know the Black-Scholes model is incorrect but still use the model to hedge and face the question of what is the appropriate value of σ_{imp} to use. If they use a value that doesn't match (in a general sense) some true "average" level of volatility then they will experience P&L profiles of the form displayed in Figure 2.



(a) Delta-hedging P&L: true vol. = 30%, imp. vol. = 20%. (b) Delta-hedging P&L: true vol. = 30%, imp. vol. = 40%.

Figure 2: Histogram of P&L from simulating 100k paths where we hedge a short call position with $S_0 = K = \$100$, $T = 6$ months, true volatility $\sigma = 30\%$, and $r = q = 1\%$. A time step of $dt = 1/2,000$ was used so hedging P&L due to discretization error is negligible. The hedge ratio, i.e. delta, was calculated using the implied volatility that was used to calculate the initial option price.

option moves significantly into or out of the money.

Returning to the self-financing trading strategy of (5) and (6), note that we can choose any model we like for the security price dynamics. In particular, we are not restricted to choosing GBM and other diffusion or jump-diffusion models could be used instead. It is interesting to simulate these alternative models and to then observe what happens to the replication error in (8) where the δ_{t_i}'s are computed assuming (incorrectly) GBM price dynamics. Note that it is common to perform simulation experiments like this when using a model to price and hedge a particular security. The 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. This back-testing is sometimes called a *historical simulation*. ∎

Simulating Multidimensional (Geometric) Brownian Motions

It is often the case that we wish to simulate a vector of geometric Brownian motion paths. In this case we again have

$$S_{t+\Delta t}^{(i)} = S_t^{(i)} e^{(\mu_i - \sigma_i^2/2)\Delta t + \sigma_i \left(B_{t+\Delta t}^{(i)} - B_t^{(i)} \right)} \tag{9}$$

for $i = 1, \ldots, n$, and where the Brownian increments $B_{t+\Delta t}^{(i)} - B_t^{(i)}$ and $B_{t+\Delta t}^{(j)} - B_t^{(j)}$ have correlation $\rho_{i,j}$. Since we know how to simulate multivariate normal distribution using the Cholesky decomposition method of Section 5.1, it should be clear how to simulate (9) for $i = 1, \ldots, n$.