Computing for Data Science
or Stochastic Computation and Algorithms

Course notes for STAT 413

Adam B Kashlak
Mathematical & Statistical Sciences
University of Alberta
Edmonton, Canada, T6G 2G1

March 26, 2023

Contents

Preface

> Never really seized by the need to calculate, he was more apt to be aware of pattern than of brute numeration.
>
> *Ratner's Star*
> Don DeLillo (1976)

This course recently came into being a few years ago. At its birth, it was called *Statistical Computing*. Then, it was rebranded as *Computing for Data Science*. If I were to give is a more descriptive name, I would call it *Stochastic Computation and Algorithms*. The main focus of these notes is how to calculate a desired number using randomization to achieve it. In Chapter 1, we first discuss how to generate random numbers using a deterministic computer. It has been said that "It may seem perverse to use a computer, that most precise and deterministic of all machines conceived by the human mind, to produce 'random' numbers" (Press et al., 2007). Nevertheless, we require random numbers, for some definition of random, in order to proceed with stochastic algorithms. In Chapter 2, we consider how to use stochastic algorithms to compute volumes and integrate functions. In Chapter 3, we consider how to use stochastic algorithms to optimize a function.

There are three textbooks I used as inspiration for these course notes: George Fishman's *Monte Carlo: concepts, algorithms, and applications* (Fishman, 2013); the famous *Numerical Recipes* by Press, Teukolsky, Vetterling, and Flannery (Press et al., 2007); and James Spall's *Introduction to stochastic search and optimization* (Spall, 2005).

Adam B Kashlak
Edmonton, Canada
Fall 2022

Chapter 1

Random Number Generation

Introduction

Type `?set.seed` into the R command prompt and start reading under the "Details" heading. It is easy to take random number generation for granted, but an incredible amount of work has gone into making sure that your simulations in R or Python or other languages are as random as possible given a deterministic computer. Many of the implemented methods within R or Python or other computing languages are based on algorithms such as Linear Feedback Shift Registers (LFSR) and Linear Congruential Generators (LCG). More details on these methods can be found in Chapter 7 of Fishman (2013) and Chapter 7 of Press et al. (2007). When discussing "random" numbers generated on a computer, they are often called psuedorandom numbers as they are deterministic but satisfy many properties of a set of truly random numbers.

For this chapter, we assume that it is possible to generate independent uniformly random binary variables. That is, for any $n \in \mathbb{N}$, we can generate $B_1, \ldots, B_n$ such that $\mathrm{P}(B_i = 1) = \mathrm{P}(B_i = 0) = 0.5$ and such that B_i and B_j are independent for all $i \neq j$. At the time of writing, pure randomness has not been achieved and one has to be cautious of the fact that such a sequence of B_i's are necessarily deterministic, but sufficiently chaotic to be considered as random.

From uniformly random bits comes uniformly random variates on the unit interval $[0, 1]$. We will denote $U_1, \ldots, U_n \overset{\text{iid}}{\sim} \text{Uniform}[0, 1]$ to be n independent uniform random variates. This implies that the cumulative distribution function (CDF) for U_i is

$$\mathrm{P}(U_i < u) = \begin{cases} 0, & u \leq 0 \\ u, & u \in (0, 1) \\ 1, & u \geq 1 \end{cases}$$

and $U_i \perp\!\!\!\perp U_j$ for $i \neq j$.[1] Generating such uniform variates can be done in R via the

[1] The symbol $\perp\!\!\!\perp$ is often used to indicate independence between two random variables.

function `runif( )`. This function directly calls compiled C code that implements the chosen random generator from `set.seed( )`.

Note that the events $U = 0$ and $U = 1$ have probability zero of occurring. In fact, in the C code underlying the R code for random number generation, if the psuedorandom uniform variate is exactly 0 or 1, the code adjusts very slightly up or down, respectively, so that `runif( )` never actually returns 0 or 1.

Remark 1.0.1. *In probability & measure theory, there is a theorem proving the existence of countably infinite independent random variables on* $\mathbb{R}$*. The technique of the proof precisely uses the same approach as the computer does. That is, (1) generate independent Bernoulli random variables. (2) use those to construct independent uniform random variables. (3) use the uniforms to construct a general collection of independent random variables. See STAT 571 notes for more details on this proof.*

1.1 Probability Integral Transform

Thus far, we have our collection of iid uniform random variates $U_1, \ldots, U_n$. However, more interesting and diverse distributions are often desirable. We will use $U_1, \ldots, U_n$ to construct new random variables with arbitrary probability distributions. The first tool we will make use of is the probability integral transform.

Theorem 1.1.1 (Probability Integral Transform)**.** *Let* $F : \mathbb{R} \to [0, 1]$ *be a strictly increasing continuous function such that* $\lim_{x\to-\infty} F(x) = 0$ *and* $\lim_{x\to\infty} F(x) = 1$*. Then,* $F^{-1} : [0, 1] \to \mathbb{R}$ *exists and if* $U \sim \text{Uniform}\,[0, 1]$*, then*

$$Z = F^{-1}(U)$$

has F *as its cumulative distribution function.*

Proof. We want the CDF for $Z = F^{-1}(U)$. That is,

$$\begin{aligned} \mathrm{P}\left(Z \le t\right) &= \mathrm{P}\left(F^{-1}(U) \le t\right) \\ &= \mathrm{P}\left(U \le F(t)\right) = F(t). \end{aligned}$$

□

Alternatively, this theorem implies that if Z has continuous increasing CDF F, then $F(Z)$ is uniform on $[0, 1]$. Note that this theorem can be extended to CDFs with discontinuities by defining the inverse CDF as follows:

$$F^{-1}(u) = \inf\{t \in \mathbb{R} \,:\, F(t) \ge u\}$$

for $u \in [0, 1]$. A jump discontinuity in a CDF implies a point mass exists in the distribution. Only a countable number of these are possible.

1.1.1 CDFs with computable inverses

If the function F has an inverse that exists and furthermore if that inverse can be expressed in a closed form analytically, then generating random variables from F is as easy as evaluating the inverse F^{-1}. However, just because generating random variables from F is easy to code does not imply that this approach is computationally efficient. More on this will be discussed in Section 1.2.

Example 1.1.1 (Exponential Distribution)**.** *The exponential distribution is critically important in queuing processes, Markov processes, and other models where a "memoryless" waiting time is required. Thus, being able to simulate exponential random variates is very desirable.*

The exponential distribution with rate parameter $\lambda > 0$ respectively has PDF and CDF

$$f(x) = \lambda \mathrm{e}^{-\lambda x} \text{ and } F(x) = 1 - \mathrm{e}^{-\lambda x}.$$

Hence, $F^{-1}(u) = -\lambda^{-1}\log(1-u)$. However, if $U \sim \text{Uniform}[0,1]$ then $1 - U \sim \text{Uniform}[0,1]$. Thus, to generate exponential random variates with rate λ, we can compute

$$X = -\frac{1}{\lambda}\log(U).$$

The parameter λ is referred to as the rate parameter as it can be interpreted as the reciprocal of the expected waiting time. That is, $\mathrm{E}X = 1/\lambda$. Hence, if we are modelling, say, the time until the next lightning strike occurs in a thunderstorm or the next atom radioactively decays, a larger λ means a faster rate of occurrences.

Note that while this is an easy way to generate exponential random variates, it is not the way that R does it with the `rexp()` function in the base `stats` package. This will be discussed in a subsequent section.

Example 1.1.2 (Cauchy Distribution)**.** *The Cauchy distribution is a Student's t-distribution with only 1 degree of freedom. While is has a bell-curve shape similar to the normal distribution, none of the moments of the Cauchy distribution are defined. e.g. it does not have a finite mean.*

A random variable X is Cauchy if it respectively has the PDF and CDF

$$f(x) = \frac{1}{\pi(1+x^2)} \text{ and } F(x) = \frac{1}{\pi}\arctan(x) + \frac{1}{2}.$$

Thus, we can write $X = F^{-1}(U) = \tan(\pi(U - 1/2))$ to generate Cauchy random variates X given $U \sim \text{Uniform}[0,1]$

The R source code uses this approach for generating Cauchy random variates via `rcauchy()`. For the more general t-distribution, one can note that if $Z \sim \mathcal{N}(0,1)$, $V \sim \chi^2(\nu)$, and $Z \perp\!\!\!\perp V$, then $X = Z/\sqrt{\nu} \sim t(\nu)$. Thus, the R code in `rt.c` generates a standard normal random variate Z and a chi squared variate V in order to return a t-random variate, which will be Cauchy if $\nu = 1$.

1.1.2 CDFs without computable inverses

If often is the case that F^{-1} exists, but no closed form analytic expression is known. The most obvious example of this is the normal or Gaussian distribution. The CDF is written in terms of an integral

$$F(x) = \frac{1}{2\pi} \int_{-\infty}^{x} \mathrm{e}^{-t^2/2} dt$$

which does not have a closed form.

However, such a issue does not completely ruin our ability to use the probability integral transform. Indeed, if we wish to compute $x = F^{-1}(u)$ for some $u \in [0, 1]$, this is equivalent to finding an x such that $0 = F(x) - u$. Hence, we have a root finding problem, which can be solved by many algorithms. Most notably, we can use the Newton-Raphson method, which is detailed in Algorithm 1.

The Newton-Raphson Method

The Newton-Raphson algorithm can be used to find the root of $F(x) - u$ assuming that the PDF $f(x) = F'(x)$ exists and that both $f(x)$ and $F(x)$ can be evaluated on a computer. The idea behind the Newton-Raphson algorithm is to start at a point x_0 in the domain $\mathcal{D}$ of F. Then, a tangent line can be drawn passing through the point $(x_0, F(x_0) - u)$. Let x_i be the root of this tangent line, then

$$f(x_0) = \frac{F(x_0) - u}{x_0 - x_1}$$

and solving for x_1 gives

$$x_1 = x_0 - \frac{F(x_0) - u}{f(x_0)}.$$

The algorithm can be iterated by repeating the above with x_1 in place of x_0 to get a new root x_2. This may continue until convergence is achieved.

It can be shown using Taylor's Theorem that if F has a continuous second derivative, then the Newton-Raphson algorithm converges quadratically. Indeed, denoting the true root by x_∞, we have that

$$|x_{i+1} - x_\infty| = C_i (x_i - x_\infty)^2.$$

But this requires some additional conditions such as $f(x_\infty) = F'(x_\infty) \neq 0$. Otherwise, convergence can be hindered.

Furthermore, the choice of starting point is very important for quick convergence. Also, certain functions will cause this algorithm to fail. For example, try applying Newton-Raphson to $F(x) = \sqrt{x}$ for $x \geq 0$ and $F(x) = -\sqrt{-x}$ for $x < 0$ for an initial value of $x_0 = 1$ and see what happens.

Algorithm 1 The Newton-Raphson Algorithm

Input:

$F : \mathcal{D} \to \mathbb{R}$,	an invertible differentiable function
$f : \mathcal{D} \to \mathbb{R}^+$,	the derivative $f = F'$
$x_0 \in \mathcal{D}$,	an initial value in the domain of F
$\tau \in \mathbb{R}^+$,	a threshold for convergence

Algorithm:

For $i \geq 1$
  $x_i = x_{i-1} - F(x_{i-1})/f(x_{i-1})$
  if $|x_i - x_{i-1}| < \tau$
    stop
return x_i

Output:

x_i such that $F(x_i) \approx 0$

Remark 1.1.3 (Halley's Method)**.** *Many other methods of root finding exist. One method known as Halley's method can be achieved by applying Newton-Raphson to the function* $F(x)/\sqrt{|F'(x)|}$ *assuming that for any* x *such that* $F(x) = 0$*, we have that* $F'(x) \neq 0$*. In Halley's method, the update set of*

$$x_{i+1} = x_i - \frac{F(x_i)}{F'(x_i)}$$

is replaced by

$$x_{i+1} = x_i - \frac{2F(x_i)F'(x_i)}{2F'(x_i)^2 - F(x_i)F''(x_i)}.$$

This approach comes with its own assumptions and concerns, which will be left to other courses to investigate. See "Householder Methods" for even more.

The Bisection Method

In the context of random variate generation, the existence of a CDF usually implies the existence of a PDF. However, given a function F to find a root, we may not have access to the derivative f. In that case, we require a gradient-free root finding method. Many of these exist. In this section, we will highlight a simple algorithm applicable when it is known that the root of F lies in the finite interval $[a, b]$ and $F(a) < 0$ and $F(b) > 0$.[2] This is the Bisection algorithm detailed in Section 2.3 of Fishman (2013) and written out in Algorithm 2. It is effectively a binary search algorithm.

[2] Or alternatively, $F(a) > 0 > F(b)$. Just as long as the signs are different.

Much like a binary search algorithm, the Bisection algorithm is guaranteed to stop after $\lceil \log_2 ((b-a)/\tau_1) \rceil$ number of steps. Hence, the tighter the interval $[a, b]$ is, the quicker the algorithm will converge, so like the starting value for Newton-Raphson, the closer you are to the true root, the faster the algorithm will terminate.

Example 1.1.4 (Beta Distribution)**.** *The beta distribution commonly occurs as a Bayesian prior and posterior distributions when a parameter of interest lies in the unit interval. If* $X \sim \text{Beta}(\alpha, \beta)$*, then the PDF is*

$$f(x) = \frac{1}{B(\alpha, \beta)} x^{\alpha-1}(1-x)^{\beta-1}$$

for $x \in [0, 1]$ *where* $B(\alpha, \beta) = \Gamma(\alpha)\Gamma(\beta)/\Gamma(\alpha+\beta)$ *is the beta function and* $\Gamma(\,)$ *is the gamma function.*[3] *The CDF is the known at the regularized incomplete beta function*

$$F(x) = \frac{1}{B(\alpha, \beta)} \int_0^x t^{\alpha-1}(1-t)^{\beta-1} dt.$$

Of note, when $\alpha = \beta = 1/2$*, the distribution is known as the arcsin distribution as its CDF is*

$$F(x) = \frac{2}{\pi} \arcsin(\sqrt{x})$$

for $x \in [0, 1]$*. Finding a root* x *of* $F(x) - u$ *for some* $u \in [0, 1]$ *could be done with the Bisection algorithm.*

1.1.3 CDFs with discrete distributions

There are many discrete probability distributions that we may wish to simulate from. These include the binomial, Poisson, geometric, and hypergeometric distributions. In such a case, we have a discrete support $\mathcal{D} = x_0, x_1, x_2, \ldots$ and a CDF F such that

$$0 = F(x_0) < F(x_1) < \ldots < F(x_i) < F(x_{i+1}) < \ldots \leq 1.$$

If the support is finite with, say, $|\mathcal{D}| = m$, then $F(x_m) = 1$. Otherwise, $F(x_i) \to 1$ as $i \to \infty$.[4]

One easy approach (coding-wise) to generating from random variates from a discrete distribution is to first compute the ordered values $F(x_i) \in [0, 1]$. Then, generate a random $U \sim \text{Uniform}[0, 1]$ and find the smallest i such that $F(x_i) < U$. In practice, the C code written to generate, say, Poisson or binomial random variables for R is immensely complex and needing to consider various extreme cases and numerical stability.

[3] For $n \in \mathbb{N}$, $\Gamma(n) = (n-1)!$ and for general $x \in \mathbb{R}^+$, $\gamma(x) = \int_0^\infty t^{x-1} \mathrm{e}^{-x} dt$.

[4] We can also extend the support of F to $-\infty$ if desired. However, most discrete distributions of interest have support contained in $\mathbb{Z}^+$.

Algorithm 2 The Bisection Algorithm

Input:

$F : \mathcal{D} \to \mathbb{R}$, an function with a root to find

$a, b \in \mathcal{D}$ such that $F(a) < 0 < F(b)$, interval to search

$\tau_1, \tau_2 \in \mathbb{R}^+$, thresholds for convergence

Algorithm:

Set $x_0 = a$ and $x_1 = b$

For $i \geq 1$

Compute $z = (x_0 + x_1)/2$

Compute $F(z)$

if $F(z) < 0$

Set $x_0 = z$

else if $F(z) \geq 0$

Set $x_1 = z$

if $|x_1 - x_0| < \tau_1$ or $|F(z)| < \tau_2$

stop

return z

Output:

z such that $F(z) \approx 0$

1.2 Other Transformations

The probability integral transform described in the previous section takes a uniform random variate U and transforms it into a new random variate with some CDF F. In this section, we will consider other ways to transform one random variate into another.

Example 1.2.1 (Student's t distribution)**.** *The t distribution is of critical importance in statistical hypothesis testing under the assumption of Gaussian errors. However, we may also wish to generate t random variates. The t distribution with ν degrees of freedom has PDF*

$$f(x) = C(1 + x^2/\nu)^{-(\nu+1)/2}$$

for some normalizing constant C and a CDF in terms of something called a Gaussian Hypergeometric Function. Suffice to say, we don't want to try to write down F let alone invert it. Luckily, as noted above in Example 1.1.2 on the Cauchy distribution, a t random variate X with ν degrees of freedom can be defined as $X = Z/\sqrt{V}$ where $Z \sim \mathcal{N}(0, 1)$, $V \sim \chi^2(\nu)$, and $Z \perp\!\!\!\perp V$. This is precisely how R generated t random variates.

Example 1.2.2 (Exponential distribution)**.** *As discussed in Example 1.1.1, exponential random variates with rate parameter $\lambda > 0$ can be generated by using the*

probability integral transform

$$X = -\frac{1}{\lambda}\log(U)$$

for $U \sim \text{Uniform}\,[0,1]$. *However, more efficient methods exist.*

In Fishman (2013) Section 3.10, it is noted that more efficient methods of generation exist. In particular, let X_0 *be a random variate from a truncated exponential distribution. That is, for* $x \in [0, \log 2]$, X_0 *has PDF and CDF*

$$f_{X_0}(x) = 2\mathrm{e}^{-x} \text{ and } F_{X_0}(x) = 2 - 2\mathrm{e}^{-x}.$$

Next, let $K \sim \text{Geometric}(1/2)$, *which is supported on the non-negative integers such that*

$$p_K(k) = \mathrm{P}\,(K = k) = (1/2)^{k+1}$$

such that $X_0 \perp\!\!\!\perp K$. *Then, we can write* $X = X_0 + K\log 2$, *which is* $\text{Exponential}\,(1)$. *Indeed, any* $x \in \mathbb{R}^+$ *can be uniquely written at* $x = x_0 + k\log 2$ *for* $k \in \mathbb{Z}^+$ *and* $x_0 \in [0, \log 2)$. *Denoting the convolution product by* $\star$, *the PDF is*

$$\begin{aligned} f_X(x) = (f_{X_0} \star p_K)(x) &= \mathrm{P}\,(K = k)\, f_{X_0}(x_0) \\ &= 2^{-k-1}2\mathrm{e}^{-x_0} = \exp(-k\log 2 - x_0) = \mathrm{e}^{-x}. \end{aligned}$$

Meanwhile, the current R source code uses an algorithm published by Ahrens & Dieter in 1972 in a paper called "Computer methods for sampling from the exponential and normal distributions."

1.2.1 Box-Muller Transform

The Gaussian or Normal distribution is perhaps the most widely used probability distribution. While its shape is an elegant bell curve, the CDF has no closed form making use of tools like the Probability Integral Transform very tedious to implement. Luckily, the Box-Muller transform gives us an alternative approach.

We say that $X \sim \mathcal{N}\left(\mu, \sigma^2\right)$ if

$$f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{1}{2}(x-\mu)^2\right)$$

for $\mu \in \mathbb{R}$ and $\sigma^2 > 0$. Strictly speaking $\sigma^2 = 0$ allows for a degenerate Gaussian distribution that reduces to a point mass at the mean μ. It can be shown that if $Z \sim \mathcal{N}\,(0,1)$, which is referred to as a standard normal distribution, that $X = \mu + \sigma Z \sim \mathcal{N}\left(\mu, \sigma^2\right)$. Hence, we only need to be concerned with generating standard normal random variates Z.

Theorem 1.2.1 (Box-Muller Transform, 1958)**.** *If* $U \sim \text{Uniform}\,[0,1]$ *and* $V \sim \text{Exponential}\,(1)$ *and* $U \perp\!\!\!\perp V$, *then*

$$\begin{aligned} x &= \sqrt{2v}\cos(2\pi u), \text{ and} \\ y &= \sqrt{2v}\sin(2\pi u) \end{aligned}$$

are independent $\mathcal{N}(0,1)$ *random variables.*

Proof. This theorem is proven via a change of variables. Note that

$$V = \frac{1}{2}(X^2 + Y^2) \text{ and}$$
$$U = \frac{1}{2\pi}\arctan(Y/X).$$

Then, defining the functions $v(x,y) = (x^2+y^2)/2$ and $u(x,y) = \arctan(y/x)/2\pi$, gives partial derivatives

$$\begin{aligned}
\frac{\partial v}{\partial x} &= x \quad \text{and} \quad \frac{\partial v}{\partial y} = y \\
\frac{\partial u}{\partial x} &= \frac{1}{2\pi}\frac{1}{1+y^2/x^2}\frac{-y}{x^2} = \frac{-y}{4\pi v(x,y)} \\
\frac{\partial u}{\partial y} &= \frac{1}{2\pi}\frac{1}{1+y^2/x^2}\frac{1}{x} = \frac{x}{4\pi v(x,y)}
\end{aligned}$$

Then, be change of variables, we have that

$$\begin{aligned}
f_{X,Y}(x,y) &= f_{U,V}(u,v)\left|\frac{\partial u}{\partial x}\frac{\partial v}{\partial y} - \frac{\partial u}{\partial y}\frac{\partial v}{\partial x}\right| \\
&= \mathbf{1}[u(x,y) \in [0,1]]\, \mathrm{e}^{-v(x,y)}\left|-\frac{y^2}{4\pi v(x,y)} - \frac{x^2}{4\pi v(x,y)}\right| \\
&= \frac{1}{2\pi}\exp\left\{-\frac{x^2+y^2}{2}\right\}.
\end{aligned}$$

As this is the Gaussian PDF in $\mathbb{R}^2$ with mean zero and covariance I_2, we have our conclusion. □

The Box-Muller transform reduces the task of generating a normal random variate to the tasks of generating a uniform and an exponential random variate. However, as is typical in these notes, more sophisticated and optimized generation methods exist in the R source code.

1.3 Acceptance-Rejection Sampling

Acceptance-Rejection sampling, sometimes just called Rejection sampling, is a powerful technique for sampling from complex distributions. In short, we wish to sample from some probability density $f(x)$, but it is hard to do so. The solution is to find a proposal or candidate distribution $h(x)$ that is close to f and easy to sample from. Then, we sample from h instead and reject any samples that don't look like they came from f. This is made more precise in the following theorem attributed to John Von Neumann.

Note that the acceptance-rejection method is a critical step in the Metropolis-Hastings algorithm for performing Markov Chain Monte Carlo. However, MCMC is beyond scope of this course. In these notes, we restrict to independent sampling rather than invoking Markov chains.

Remark 1.3.1. *In the following theorem, we use | to denote conditioning. That is, we define a random variable X to equal another random variable Z after forcing a condition to hold. For example, you could consider a random 20-sided dice roll conditioned on the number displayed is less than 8.*

In general, for two real-valued random variables X and Y, we can write the conditional density function as

$$f_{Y|X}(y|x) = \frac{f(x,y)}{f_X(x)}$$

where $f(x,y)$ is the joint density function.

Theorem 1.3.1 (Von Neumann, 1951)**.** *Let $f(x)$ be a pdf with support on $[a,b]$ that can be written as $f(x) = cg(x)h(x)$ where $h(x) \geq 0$, $\int_a^b h(x)dx = 1$, $g(x) \in [0,1]$, and some constant $c \geq 1$. If $Z \sim h(z)$ and $U \sim \text{Uniform}\,[0,1]$ and $Z \perp\!\!\!\perp U$, then $X = Z|\{U \leq g(Z)\}$ has pdf $f(x)$.*

Proof. Since U and Z are independent, the joint pdf is simply $f(u,z) = \mathbf{1}_{u\in[0,1]}h(z)$. Consequently, the conditional pdf for X is

$$h(z|U \leq g(Z))\text{P}\left(U \leq g(Z)\right) = \int_0^{g(z)} f(u,z)du$$
$$h(z|U \leq g(Z)) = \frac{\int_0^{g(z)} f(u,z)du}{\text{P}\left(U \leq g(Z)\right)}.$$

The numerator becomes $\int_0^{g(z)} f(u,z)du = g(z)h(z)$. The denominator becomes

$$\begin{aligned}\text{P}\left(U \leq g(Z)\right) &= \iint_{[0,1]\times[a,b]} \mathbf{1}[u \leq g(z)]\, du\, h(z)dz \\ &= \int_a^b \int_0^1 \mathbf{1}[u \leq g(z)]\, du\, h(z)dz = \int_a^b g(z)h(z)dz\end{aligned}$$

As $f(x)$ is a pdf,

$$1 = \int_a^b f(x)dx = c\int_a^b g(x)h(x)dx.$$

Thus, $\int_a^b g(x)h(x)dx = c^{-1}$, and consequently, $h(z|U \leq g(Z)) = f(z)$. □

Remark 1.3.2. *While many choices of c and $g(x)$ are possible, the acceptance-rejection method is most efficient when c is close to 1. This is because we accept a*

Algorithm 3 The Acceptance-Rejection Algorithm

Input:
$g(x) \in [0, 1]$
$h(x)$, a pdf to sample from
Algorithm:
Repeat
Generate z from pdf $h(z)$
Generate u from Uniform $[0, 1]$
if $u \leq g(z)$
stop
return $x = z$
Output:
x from pdf $f(x) = cg(x)h(x)$.

randomly generated variate X when $U \leq g(Z)$ which occurs with probability $1/c$ as per the above proof.

Hence, given a pdf of interest f and a candidate distribution h, then c is chosen to be

$$c = \sup_x \{f(x)/h(x)\}$$

and g chosen accordingly.

Furthermore, we can consider the event $U \leq g(Z)$ to have a geometric distribution with probability $1/c$ of success. Hence, the expected number of iterates until acceptance is achieved is c.

Example 1.3.3 (Half-Normal Distribution)**.** *This example is taken from Example 3.1 of Fishman (2013). The half-normal distribution is the standard normal but only supported on the non-negative reals. That is,*

$$\begin{aligned} f(x) &= \sqrt{\frac{2}{\pi}} \mathrm{e}^{-x^2/2} \\ &= \sqrt{\frac{2\mathrm{e}}{\pi}}\, \mathrm{e}^{-(x-1)^2/2}\, \mathrm{e}^{-x}. \end{aligned}$$

Thus, taking $c = \sqrt{2\mathrm{e}/\pi}$, $g(x) = \exp[-(x-1)^2/2]$, and $h(x) = \exp(-x)$, we can sample from the exponential distribution e^{-x} to apply the Algorithm 3 to generate half-normal random variates.

Furthermore, this allows for an alternative way to sample from the normal distribution. That is, use Algorithm 3 to generate X as half-normal. Then, generate $U \sim$ Uniform $[0, 1]$ and set

$$X \leftarrow \begin{cases} X & \text{if } U \geq 1/2 \\ -X & \text{if } U < 1/2 \end{cases}.$$

Algorithm 4 The Acceptance-Rejection-Squeeze Algorithm

Input:
$g, g_L, g_U \in [0,1]$ such that $g_L \leq g \leq g_U$
$h(x)$, a pdf to sample from
Algorithm:
Repeat
Generate z from pdf $h(z)$
Generate u from Uniform $[0,1]$
if $u \leq g_L(z)$ *pretest with* g_L
stop
else if $u \leq g_U(z)$ *pretest with* g_U
if $u \leq g(z)$ *only evaluate* g *if pretests fail*
stop
return $x = z$
Output:
x from pdf $f(x) = cg(x)h(x)$.

In this setting, we require one Uniform and one Exponential random variate until a sample is "accepted". Then, one final Uniform random variate is required to set the sign, i.e. positive or negative.

1.3.1 Acceptance-Rejection-Squeeze

Sometimes the function $g(x)$ may be slow to compute. However, if we can construct an envelope $g_L(x) \leq g(x) \leq g_U(x)$ where g_L, g_U are "simpler" functions, then we have use g_L and g_U to perform pretests on whether or not we should accept or reject.

The most natural choices for g_L and g_U come from power series. For example, $1 - x \leq \mathrm{e}^{-x} \leq 1 - x + x^2/2$ for $x \geq 0$. Hence, instead of merely evaluating $\exp[-(x-1)^2/2]$ in the previous example for the half-normal distribution, we could choose

$$g_L(x) = 1 - \frac{(x-1)^2}{2} \text{ and } g_U(x) = 1 - \frac{(x-1)^2}{2} + \frac{(x-1)^4}{8}.$$

1.3.2 Box-Muller without Trigonometry

Instead of generating two independent uniform deviates to perform the Box-Muller transform (i.e. sampling within the square $[0,1] \times [0,1]$), we can instead sample from within the unit circle using an acceptance-rejection step to generate normal random variates without need to evaluate sines and cosines.

Indeed, we can generate $U, V \sim$ Uniform $[-1,1]$ until $U^2 + V^2 \leq 1$ and hence lies within the unit circle. This means that the ordered pair (U, V) has a uni-

formly random angle between 0 and 2π and a distance to the origin (radius) that is Uniform $[0,1]$. After that is achieved, we claim that

$$\sqrt{-2\log(U^2+V^2)\frac{V^2}{U^2+V^2}} \sim \mathcal{N}(0,1).$$

This is because U^2+V^2 is Uniform $[0,1]$ thus making $-\log(U^2+V^2) \sim$ Exponential (1). Furthermore, the sine and cosine of the angle of the ordered pair (U,V) can be computed as $\frac{U}{\sqrt{U^2+V^2}}$ and $\frac{V}{\sqrt{U^2+V^2}}$. Thus, we can compare to the original Box-Muller transform

$$x = \sqrt{2v}\cos(2\pi u), \text{ and}$$
$$y = \sqrt{2v}\sin(2\pi u)$$

for u uniform and v exponential in this case.

The same trick can be applied generate Cauchy random variates. Recall from Example 1.1.2 that for $U \sim$ Uniform $[0,1]$, then $X = \tan[\pi(U-1/2)]$ is Cauchy. If we preferred to not evaluate the tangent, we can instead sample from the upper-half of the unit circle. That is, generate $U \sim$ Uniform $[-1,1]$ and $V \sim$ Uniform $[0,1]$ until $U^2+V^2 \leq 1$. Then, the tangent of the angle of the ordered pair (U,V) is simply

$$U/V \sim \text{Cauchy}.$$

See sections 7.3.4 and 7.3.7 of Press et al. (2007) for detailed code for these methods.

1.4 Ratio of Uniforms

In this section, we generalize the previously discussed idea of generating a uniform element of the unit circle to simulate from more complex univariate distributions. This Ratio-of-Uniforms method is attributed to Kinderman & Monahan (Kinderman and Monahan, 1977). The idea is to define a region in $\mathbb{R}^2$ (e.g. the unit circle), then generate a uniformly random point (U,V) from that region, and then return U/V, which will have the desired distribution. Once again, we have a theorem to make this more precise.

Theorem 1.4.1 (Ratio of Uniforms)**.** *Let $f(z)$ be a pdf on $\mathbb{R}$ and $r(z)$ a non-negative function such that*

$$f(z) = \frac{r(z)}{\int_{-\infty}^{\infty} r(t)dt}.$$

Let $\mathcal{D}_0 \subset \mathbb{R}^2$ be defined as

$$\mathcal{D}_0 = \left\{(x,y) \,:\, 0 \leq x \leq \sqrt{r(y/x)}\right\}$$

and let (X,Y) be a uniform random point in $\mathcal{D}_1$, some bounded region such that $\mathcal{D}_0 \subset \mathcal{D}_1$ and let $W = X^2$. Then, if $W \leq r(Y/X)$, the $Z = Y/X$ has pdf $f(z)$.

Proof. Let $|\mathcal{D}_0|$ denote the *area*[5] of the region $\mathcal{D}_0$. Then, conditioning on (X, Y) being in $\mathcal{D}_0$, the joint pdf of (X, Y) is simply $f(x, y) = |\mathcal{D}_0|^{-1}$. Since $W = X^2$ and $Z = Y/X$, we have that $X = \sqrt{W}$ and $Y = Z\sqrt{W}$. Applying the change of variables $(X, Y) \to (W, Z)$ gives

$$f_{W,Z}(w, z) = J f_{X,Y}(\sqrt{w}, z\sqrt{w})$$

where J is the Jacobian determinant

$$J = \det \begin{pmatrix} \frac{\partial x}{\partial w} & \frac{\partial x}{\partial z} \\ \frac{\partial y}{\partial w} & \frac{\partial y}{\partial z} \end{pmatrix} = \det \begin{pmatrix} 1/2\sqrt{w} & 0 \\ z/2\sqrt{w} & \sqrt{w} \end{pmatrix} = \frac{1}{2}.$$

Therefore, $f_{W,Z}(w, z) = [2|\mathcal{D}_0|]^{-1}\mathbf{1}[0 \le w \le r(z)]$. Therefore, the pdf for Z is

$$f_Z(z) = \int_0^{r(z)} f_{W,Z}(w, z)dw = \frac{r(z)}{2|\mathcal{D}_0|}.$$

Since $f_Z(z)$ must integrate to 1, we have that $2|\mathcal{D}_0| = \int_{-\infty}^{\infty} r(z)dz$. □

The region $\mathcal{D}_0$ may be hard to visualize and even harder to sample from. Hence, we can bound $\mathcal{D}_0$ with a rectangle $\mathcal{D}_1$ by noting the following. On the *x-axis*, the region $\mathcal{D}_0$ is constrained to the interval $[0, x_\vee]$ where $x_\vee = \sup_z \sqrt{r(z)}$. On the *y-axis*, we have that $y = zx = z\sqrt{r(z)}$. Hence, the region $\mathcal{D}_0$ is constrained to the interval $[y_\wedge, y_\vee][\inf_z z\sqrt{rz}, \sup_z z\sqrt{rz}]$. Thus, we can use this rectangle for implementing the ratio of uniforms method as detailed in Algorithm 5.

Note that since we are sampling from a rectangle $\mathcal{D}_1$ rather than the actual set of interest $\mathcal{D}_0$, there is an acceptance-rejection step in Algorithm 5. Thus, the probability of accepting a randomly drawn point (X, Y) is $|\mathcal{D}_0|/|\mathcal{D}_1|$. From the above proof, this probability can be written down explicitly:

$$\mathrm{P}\left((X, Y) \in \mathcal{D}_0\right) = \frac{|\mathcal{D}_0|}{|\mathcal{D}_1|} = \frac{1}{2x_\vee(y_\vee - y_\wedge)} \int_{-\infty}^{\infty} r(z)dz.$$

Remark 1.4.1 (Shift and Scale)**.** *Note that we can always take Y/X and shift and scale it to get $a_0 + a_1(Y/X)$ if desired. The above theorem, proof, and algorithm can be modified accordingly.*

As noted, we already saw specific examples of the ratio of uniforms method. In the case of generating Cauchy random variables, we found that for $X \sim \mathrm{Uniform}\,[0, 1]$ and $Y \sim \mathrm{Uniform}\,[-1, -1]$, then $Z = Y/X$ is Cauchy conditioned on $X^2 + Y^2 \le 1$. If we set $r(z) = (1 + z^2)^{-1}$, then

$$x_\vee = \sup_{z \in \mathbb{R}} \frac{1}{\sqrt{1 + z^2}} = 1$$
$$y_\vee = \sup_{z \in \mathbb{R}} \frac{z}{\sqrt{1 + z^2}} = \sup_{z \in \mathbb{R}} \frac{z}{|z|} \frac{1}{\sqrt{1 + z^{-2}}} = 1$$

[5] i.e. Lebesgue measure

Algorithm 5 The Ratio of Uniforms Algorithm

Input:
function $r(z) = cf(z)$ for some $c > 0$
Algorithm:
Compute $x_\vee = \sup_z \sqrt{r(z)}$.
Compute $y_\vee = \sup_z z\sqrt{rz}$.
Compute $y_\wedge = \inf_z z\sqrt{rz}$.
Repeat
Generate x from pdf Uniform $[0, x_\vee]$
Generate y from pdf Uniform $[y_\wedge, y_\vee]$
if $x^2 \leq r(y/x)$
stop
return $z = y/x$
Output:
z from pdf $f(z)$.

and $y_\wedge = -1$ from the same formula. Furthermore, the half circle is recovered by noting that

$$0 \leq x \leq \frac{1}{\sqrt{1 + y^2/x^2}} \quad \Rightarrow \quad 0 \leq \sqrt{x^2 + y^2} \leq 1$$

And thus we can rederive the formula for a Cauchy random variable via the ratio of uniforms method. In this case, the probability of acceptance is $\pi/4$.

1.4.1 Gamma Random Variables

In this section, we examine the method of Cheng and Feast (1980) for generating Gamma random variates making use of the Ratio of Uniforms algorithm. Note that the current R source code uses methods attributed to Ahrens & Dieter in `rgamma()` and other methods also exist.

We say that a random variable $Z \sim \text{Gamma}(\alpha, \beta)$ for $\alpha, \beta > 0$ if $Z \geq 0$ and has pdf

$$f(z) = \frac{\beta^\alpha}{\Gamma(\alpha)} z^{\alpha-1} \mathrm{e}^{-\beta z}.$$

But if $Z \sim \text{Gamma}(\alpha, 1)$, then $\beta Z \sim \text{Gamma}(\alpha, \beta)$. Thus, we can restrict our investigate to generating Gamma random variables with $\beta = 1$.

Gamma distributions "add" in the sense that if $Z_1 \sim \text{Gamma}(\alpha_1, \beta)$ and $Z_2 \sim \text{Gamma}(\alpha_2, \beta)$ and $Z_1 \perp\!\!\!\perp Z_2$, then $Z_1 + Z_2 \sim \text{Gamma}(\alpha_1 + \alpha_2, \beta)$. Furthermore, $\text{Gamma}(1, 1)$ is the standard exponential distribution. Hence, one *could* generate a $\text{Gamma}(n, 1)$ random variable by summing n independent $\text{Exponential}(1)$ random

variables.[6] However, this would be highly computationally inefficient.

Instead, we can apply the Ratio of Uniforms method by choosing $r(z) = z^{\alpha-1}\mathrm{e}^{z}$ where we impose that $\alpha > 1$.[7] Thus, the region $\mathcal{D}_0$ is

$$\mathcal{D}_0 = \left\{(x,y) \in \mathbb{R}^2 \,:\, 0 \le x \le \sqrt{(y/x)^{\alpha-1}\mathrm{e}^{-y/x}}\right\},$$

and the bounding rectangle is

$$x \in \left[0, \left(\frac{\alpha-1}{\mathrm{e}}\right)^{(\alpha-1)/2}\right], \quad y \in \left[0, \left(\frac{\alpha+1}{\mathrm{e}}\right)^{(\alpha+1)/2}\right],$$

and the probability of an acceptance is

$$\frac{|\mathcal{D}_0|}{|\mathcal{D}_1|} = \frac{\Gamma(\alpha)\mathrm{e}^{\alpha}}{2(\alpha-1)^{\alpha-1}(\alpha+1)^{\alpha+1}}.$$

For large α, this is approximately $O(\alpha^{-1/2})$ as can be derived via Stirling's approximation to the Gamma function. This implies that the ratio of uniforms will have vanishingly small acceptance probabilities as α grows. The proposed solution to this problem is to not sample from the rectangle $\mathcal{D}_1$ but to instead construct the parallelogram defined by

$$\mathcal{P} = \left\{(x,y) \in \mathbb{R}^2 \,:\, \alpha^{-1/2} \le y - x \le \sqrt{2/\alpha\mathrm{e}},\ 0 \le y \le 1\right\}$$

and sample from $\mathcal{P} \cap [0,1] \times [0,1]$. More details on this can be found in Fishman (2013) and Cheng and Feast (1980).

1.5 Points in Geometric Objects

The following two subsections consider random generation of points within a geometric object. The first object of interest is the simplex, which has direct application to topics like Bayesian priors on probability vectors. The second is spheres and ellipsoids, which arise in areas like directional data among others.

1.5.1 Simplexes and the Dirichlet Distribution

A simplex in $\mathbb{R}^n$ is a convex polytope consisting of $n+1$ verticies. Effectively, it is a generalized "triangle" or "tetrahedron" in n-dimensions. Fixing one of the vertices at the origin allows us to define a standard simplex as

$$\mathcal{S}_n(r) = \left\{\mathbf{x} = (x_1, \ldots, x_n)^{\mathrm{T}} \,:\, \sum_{i=1}^{n} x_i \le r,\ x_i \ge 0\, \forall i\right\}$$

[6] When $\alpha = n$ is an integer, the Gamma distribution is sometimes called the Erlang distribution named after Agner Krarup Erlang.

[7] The $0 < \alpha < 1$ is handled by other algorithms.

for some constant $r > 0$. The most common setting is $r = 1$. Then, choosing $x_0 = 1 - \sum_{i=1}^{n} x_i$ gives the $(n+1)$-long vector $(x_0, x_1, \ldots, x_n)$, which is a probability vector.[8]

To generate a uniform random point inside the simplex $\mathcal{S}_n(1)$, we first generate n Uniform $[0,1]$ random variables $U_1, \ldots, U_n$ and sort them so that $U_1 \leq U_2 \leq \ldots \leq U_n$. Note that there are effcient algorithms for generating an ordered set of independent random variables; see *order statistics*. Then, we simply set $X_1 = U_1$ and $X_i = U_i - U_{i-1}$ for $i = 2, \ldots, n$. The remainder is $X_0 = 1 - U_n$.

In Bayesian analysis and other topics in statistics and machline learning, we often want to generate a probability vector $\mathbf{p} = (p_1, \ldots, p_n)$ such that $p_i \geq 0$ for all i and $\sum_{i=1}^{n} p_i = 1$. Furthermore, we may want a distribution on $\mathbf{p}$ other than the uniform. That is, we say that $\mathbf{p}$ has a Dirichlet distribution with parameter vector $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$ with $\alpha_i > 0$ for all i if it has pdf

$$f(\mathbf{p}) = \frac{\Gamma(\alpha_1 + \ldots + \alpha_n)}{\Gamma(\alpha_1) \ldots \Gamma(\alpha_n)} \prod_{i=1}^{n} p_i^{\alpha_i - 1}.$$

If all of the $\alpha_i = 1$, then we have a uniform distribution on the simplex. If the $\alpha_i > 1$, the the mass of the distribution is pushed towards the centre of the simplex. If the $\alpha_i < 1$, then the mass is pushed towards the verticies of the simplex.

An easy to code method for generating random variates from a Dirichlet distribution is to generate n independent Gamma random variables

$$X_i \sim \text{Gamma}\,(\alpha_i, 1) \;\text{ for } i = 1, \ldots, n$$

such that $X_i \perp\!\!\!\perp X_j$ for all $i \neq j$. Then, we set

$$p_i = \frac{X_i}{\sum_{i=1}^{n} X_i}.$$

This approach can be shown to yield a Dirichlet random variate, by first starting with the joint pdf of $X_1, \ldots, X_n$, which is

$$f_X(x_1, \ldots, x_n) = \prod_{i=1}^{n} \frac{x_i^{\alpha_i - 1} \mathrm{e}^{-x_i}}{\Gamma(\alpha_i)} = \exp\left(-\sum_{i=1}^{n} x_i\right) \prod_{i=1}^{n} \frac{x_i^{\alpha_i - 1}}{\Gamma(\alpha_i)}.$$

Then, we denote the sum $\sum_{i=1}^{n} x_i = T$, and we change variables from x_i to $p_i = x_i/T$ for $i = 1, \ldots, n-1$ and set $p_n = \sum_{i=1}^{n} x_i = T$. Hence, $x_i = p_i T$ for $i = 1, \ldots, n-1$, and $x_n = T(1 - \sum_{i=1}^{n-1} p_i)$. The Jacobian matrix with i, jth entry $\partial x_i / \partial p_j$ is

$$J = \begin{pmatrix} T & 0 & \ldots & p_1 \\ 0 & T & \ldots & p_2 \\ \vdots & \vdots & \ddots & \vdots \\ -T & -T & \ldots & 1 - \sum_{i=1}^{n-1} p_i \end{pmatrix}$$

[8] i.e. the entries are non-negative and sum to one.

where we can find that $\det(J) = T^{n-1}$. Thus, we have that

$$\begin{aligned} f_p(p_1, \ldots, p_{n-1}, T) &= \mathrm{e}^{-T} \frac{[T(1 - \sum_{i=1}^{n-1} p_i)]^{\alpha_n - 1} \prod_{i=1}^{n-1} (T p_i)^{\alpha_i - 1}}{\prod_{i=1}^{n} \Gamma(\alpha_i)} T^{n-1} \\ &= \frac{(1 - \sum_{i=1}^{n-1} p_i)^{\alpha_n - 1} \prod_{i=1}^{n-1} p_i^{\alpha_i - 1}}{\prod_{i=1}^{n} \Gamma(\alpha_i)} T^{\sum \alpha_i - 1} \mathrm{e}^{-T}. \end{aligned}$$

And integrating out T from 0 to ∞ gives the desired formula as

$$\Gamma(\textstyle\sum \alpha_i) = \displaystyle\int_0^\infty T^{\sum \alpha_i - 1} \mathrm{e}^{-T} dT.$$

1.5.2 Spheres

The first problem in this section is to generate a uniformly random point on the surface of an n-dimensional hypersphere (or just the usual 3D sphere, if you prefer), which is $\mathcal{S}_{n-1}(r) = \{z \in \mathbb{R}^n : \sum z_i^2 = r^2\}$. To achieve this, we can simulate a multivariate standard normal random vector and normalize it. That is, if

$$X = (X_1, \ldots, X_n) \sim \mathcal{N}(0, I_n),$$

then the vector Z with entries

$$Z_1 = \frac{X_1}{\sqrt{\sum_{i=1}^n X_i}}, \ldots, Z_n = \frac{X_n}{\sqrt{\sum_{i=1}^n X_i}},$$

will be uniformly distributed on the surface of a n-dimensional sphere of radius 1. To change the radius, multiply each Z_i by some $r > 0$.

The second problem is to generate points within an n-dimensional sphere, which is $\mathcal{B}_{n-1}(r) = \{z \in \mathbb{R}^n : \sum z_i^2 \leq r^2\}$. This can be achieved by sampling from the surface as above and the beta distribution. Indeed, let Z be uniformly random on the surface of the sphere and let

$$W \sim \mathrm{Beta}(n, 1),$$

be independent of Z. Then we claim that WZ is uniformly distributed within the sphere of radius 1 where Z is the position on the sphere and W acts as a random radius. To see this, note that

$$\begin{aligned} f_Z(z|r) &= \frac{\Gamma(n/2)}{2\pi^{n/2} r^{n-1}} && \text{for a sphere with radius } r \\ f_W(w) &= n w^{n-1} && \text{for } w \in [0, 1]. \\ f_Z(z|r = w) f_W(w) &= \frac{n\Gamma(n/2)}{2\pi^{n/2}} = \frac{\Gamma(n/2 + 1)}{\pi^{n/2}}. \end{aligned}$$

Chapter 2

Volume Computation and Integration

Introduction

In Chapter 2 of these lecture notes, we are concerned with using randomly generated variates to actually compute things. This will take on different forms. One will be to compute the volume of some region in space. Another will be to numerically integrate some function. Much as before, we will have to be clever to sample random variates in a "smart" way in order to produce efficient algorithms.

One of the new considerations in this chapter is the accuracy of computation. That is, we will be concerned with topics like error estimation, confidence intervals for estimates, and dependecy on sample size. That is, how large of a sample do we need to get an estimate with a given accuracy.

2.1 Volume Computation

Consider the following problem: Let $\mathcal{D}$ be a be a bounded region in $\mathbb{R}^d$. How can we compute the d-dimensional volume of the region $\mathcal{D}$? While the region $\mathcal{D}$ may be complex, we assume there exists a function

$$\phi(x) = \begin{cases} 1, & x \in \mathcal{D} \\ 0, & x \in \mathcal{D} \end{cases}$$

that is easy to compute. Then, we want to compute the integral

$$\text{vol}(\mathcal{D}) = \int_{\mathcal{D}} dx = \int_{\mathbb{R}^d} \phi(x) dx.$$

Hence, we will try to sample points in $\mathbb{R}^d$ to estimate the integral of $\phi(x)$.

Remark 2.1.1. *Without loss of generality, we will assume that $\mathcal{D}$ lives within the d-dimensional unit cube*

$$\mathcal{C} = [0,1]^{\otimes d}.$$

We can always scale and translate a region $\mathcal{D}$ to make this so.

2.1.1 Sampling from a regular grid

Eschewing randomness for the moment, we could merely sample from a regular grid within the unit cube $\mathcal{C}$. That is, for some $k \in \mathbb{N}$, we can generate a grid of $n = k^d$ points of the form $x = (x_1, \ldots, x_d)$ where each

$$x_i \in \left\{ \frac{0+0.5}{k}, \frac{1+0.5}{k}, \ldots \frac{(k-1)+0.5}{k} \right\}.$$

Then, we can simply count how many of these points x are in $\mathcal{D}$, i.e. $\phi(x) = 1$. This is performed in Algorithm 6. The obvious problem here is that k^d will get very big very fast as the dimension increases.

In Section 2.1 of Fishman (2013), it is claimed that the error of this estimate can be upper bounded as follows:

$$\left| \text{Vol}(\mathcal{D}) - \frac{S}{k^d} \right| \leq \frac{\text{Surface}(\mathcal{D})}{k}$$

where Surface($\mathcal{D}$) is the "surface area" of the boundary of $\mathcal{D}$. If we are in 2-dimensions, this is the length of the boundary. If we are in 3-dimensions, this is the usually idea of surface area. In d-dimensional space, this is a $(d-1)$-dimensional boundary measure. Since the number of points considered $n = k^d$, this implies that the error bound shrinks like $O(n^{-1/d})$. Another way to think about this is that if we want an error less than some $\varepsilon > 0$, then we require on the order of $n \gtrsim (1/\varepsilon)^d$ where the notation $\gtrsim$ means $n \geq C(1/\varepsilon)^d$ for some unspecified constant $C > 0$.

Example 2.1.2 (Fraction of a Sphere)**.** *As an example, consider the region*

$$\mathcal{D} = \{x \in \mathbb{R}^d \,:\, x_i \geq 0 \text{ and } x_1^2 + \ldots + x_k^2 \leq 1\}.$$

This region is $1/2^d$ of the d-dimensional sphere. By looking up formulae for volume and surface area, we find that

$$\text{Vol}(\mathcal{D}) = \frac{(\pi/4)^{d/2}}{\Gamma(d/2+1)} \quad \text{and} \quad \text{Surf}(\mathcal{D}) = \frac{2(\pi/4)^{d/2}}{\Gamma(d/2)}.$$

Therefore, we may want to compute this volume using Algorithm 6 to an accuracy of ε. However, $\text{Vol}(\mathcal{D}) \to 0$ as $d \to \infty$. Hence, we would have to adapt ε to the dimension of the problem. Therefore, we will consider the normalized error bound:

$$\frac{\left|\text{Vol}(\mathcal{D}) - k^{-d}S\right|}{\text{Vol}(\mathcal{D})} \leq \frac{1}{k} \frac{\text{Surf}(\mathcal{D})}{\text{Vol}(\mathcal{D})}$$

Algorithm 6 Volume via regular grid

Input:

function $\phi(x)$ that is 1 on $\mathcal{D}$ and 0 otherwise.

$k \in \mathbb{N}$, the number of points per dimension to sample.

Algorithm:

Set $S = 0$.

Construct a grid $\mathcal{X}$ of k^d points x.

for each $x \in \mathcal{X}$

  if $\phi(x) = 1$

    $S \leftarrow S + 1$

return S/k^d

Output:

Volume estimate of region $\mathcal{D}$.

Then, the sample size becomes

$$\begin{aligned}
\varepsilon &= n^{-1/d} \frac{\text{Surf}(\mathcal{D})}{\text{Vol}(\mathcal{D})} \\
n &= \varepsilon^{-d} \left(\frac{\text{Surf}(\mathcal{D})}{\text{Vol}(\mathcal{D})} \right)^d \\
&= \left(\frac{2}{\varepsilon} \right)^d \left(\frac{\Gamma(d/2+1)}{\Gamma(d/2)} \right)^d \\
&= \left(\frac{d}{\varepsilon} \right)^d
\end{aligned}$$

Hence, if we want the **relative** *error less than, say,* 0.001*, then the sample sizes are approximately*

$$10^{6.6},\ 10^{10.4},\ 10^{14.4},\ 10^{18.5}$$

for $d = 2, 3, 4, 5$.

2.1.2 Uniform Monte Carlo

Instead of using a regular grid, we can do what we did in Chapter 1 of these notes. That is, we can sample uniformly at random from the unit cube $\mathcal{C}$ and use acceptance-rejection to compute the volume of $\mathcal{D}$.

Indeed, we can simulate $X_1, \ldots, X_n \in \mathcal{C}$ with iid Uniform $[0, 1]$ entries. Then,

$$\text{vol}(\mathcal{D}) = \int_{\mathbb{R}^d} \phi(x) dx = \int_{\mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} \phi(x_i) dx \approx \frac{1}{n} \sum_{i=1}^{n} \phi(x_i) = \frac{|\{x_i \in \mathcal{D}\}|}{n}.$$

This algorithm is detailed in Algorithm 7. Going one step farther, we can compute the variance of this estimate as well. We can consider each $b_i = \phi(x_i)$ to be a Bernouli random variable with probability $|\mathcal{D}|$ of getting a 1 and $1 - |\mathcal{D}|$ of getting a zero. Hence, for each i

$$\operatorname{Var}(b_i) = |\mathcal{D}|(1 - |\mathcal{D}|),$$

and for our estimate, we have

$$\sigma^2 = \operatorname{Var}\left(\frac{1}{n}\sum_{i=1}^{n} b_i\right) = \frac{|\mathcal{D}|(1 - |\mathcal{D}|)}{n}.$$

Thus, the variance of the estimate decreases at the usual n^{-1} rate. We can also compute the unbiased estimate of the variance, which is

$$\hat{\sigma}^2 = \frac{(S/n)(1 - S/n)}{n - 1}.$$

The "usual" normal style confidence interval is

$$\left||\mathcal{D}| - \frac{S}{n}\right| \leq t_{1-\alpha/2,n-1}\hat{\sigma}$$

where $t_{1-\alpha/2,n-1}$ is the value such that $\mathrm{P}\left(T > t_{1-\alpha/2,n-1}\right) = \alpha/2$ where $T \sim t\,(n-1)$. This hints that our estimation error decreases at a rate of $O(n^{-1/2})$ for sample size n.

The problem with this is that in the normal distribution case, the estimate for the mean $\hat{\mu}$ and the estimate for the variance $\hat{\sigma}^2$ are independent of each other. However, for these Bernouli random variables, this is not the case. Hence, S/n and $\hat{\sigma}^2$ may be correlated, which can invalidate the confidence interval. We will consider two more powerful methods for confidence interval construction and sample size estimation.

Chebyshev's Inequality

One approach to confidence interval construction is to use Chebyshev's inequality. This is a very general and very useful upper bound on the tail probability of a random variable only assuming a finite variance. Hence, it can apply to a wide range of distributions.

Theorem 2.1.1 (Chebyshev's Inequality)**.** *Let $Z \in \mathbb{R}$ be a mean zero random variable with $\mathrm{E}Z^2 < \infty$. Then,*

$$\mathrm{P}\left(|Z| > \varepsilon\right) \leq \frac{\mathrm{E}Z^2}{\varepsilon^2}.$$

Algorithm 7 Volume via uniform Monte Carlo

Input:

function $\phi(x)$ that is 1 on $\mathcal{D}$ and 0 otherwise.

$n \in \mathbb{N}$, the number random points to simulate.

Algorithm:

Set $S = 0$.

for $i = 1, \ldots, n$

Generate x with iid entries $x_j \sim \text{Uniform}\,[0,1]$ for $j = 1, \ldots, d$.

if $\phi(x) = 1$

$S \leftarrow S + 1$

return estimate S/n

return variance $[(S/n)(1 - S/n)]/(n-1)$

return confidence interal (see below)

Output:

Volume estimate of region $\mathcal{D}$ along with variance and confidence interval.

As computed above, we have $Z = S/n - |\mathcal{D}|$. Then, $\mathrm{E}Z^2$ is the variance of the estimate being $|\mathcal{D}|(1-|\mathcal{D}|)/n$. Applying Chebyshev's inequality results in

$$\mathrm{P}\left(|S/n - \mathcal{D}| > \varepsilon\right) \leq \frac{|\mathcal{D}|(1-|\mathcal{D}|)}{n\varepsilon^2} \leq \frac{1}{4n\varepsilon^2}.$$

This is because $|\mathcal{D}|(1-|\mathcal{D}|) \leq 1/4$. Therefore, if we want a confidence level of $1-\alpha$, then we set

$$\alpha = (4n\varepsilon^2)^{-1} \quad \Rightarrow \quad \varepsilon = (4n\alpha)^{-1/2}$$

and conclude that the confidence interval

$$|S/n - \mathcal{D}| \leq (4n\alpha)^{-1/2}$$

has probability at least $1-\alpha$.

Using this approach, we learn a few things. First, we see again that the error tolerance $\varepsilon = O(n^{-1/2})$ for a fixed confidence level $1-\alpha$. We also see that the sample size $n = (2\alpha\varepsilon^2)^{-1}$. Hence, n is inversely proportional to α; e.g. cutting α in half requires a doubling of the sample size. Lastly, even though we have the same $O(n^{-1/2})$ conclusion as occurs for the normal distribution, usage of Chebyshev's inequality does not invoke the Central Limit Theorem and hence applies for any n as opposed to as $n \to \infty$.

Hoeffding's Inequality

Hoeffding's Inequality is a slightly more sophisticated result than Chebyshev's inequality. It applies to sums if bounded random variables[1] such as sums of Bernoulli

[1] and more generally to sums of sub-Gaussian random variables

random variables, which is how we state the following version of Hoeffding's inequality.

Theorem 2.1.2 (Hoeffding's Inequality)**.** *Let* $B_1, \ldots, B_n \overset{iid}{\sim} \text{Bernoulli}\,(p)$ *for some* $p \in (0,1)$ *and* $\bar{B} = n^{-1}\sum_{i=1}^{n} B_i$*. Then,*

$$\mathrm{P}\left(|\bar{B} - p| > t\right) \leq 2\exp(-2nt^2).$$

Applying Hoeffding's inequality to the above problem results in

$$\mathrm{P}\left(|S/n - |\mathcal{D}|| > \varepsilon\right) \leq 2\exp(-2n\varepsilon^2).$$

Therefore, a $1-\alpha$ confidence interval can be computed by

$$\alpha = 2\mathrm{e}^{-2n\varepsilon^2} \quad \Rightarrow \quad \varepsilon = \sqrt{-\frac{\log(\alpha/2)}{2n}},$$

which results in the confidence interval

$$|S/n - |\mathcal{D}|| \leq \sqrt{-\frac{\log(\alpha/2)}{2n}}$$

having probability at least $1-\alpha$.

Once again, we see that the width of the confidence interval is proportional to $n^{-1/2}$ much like with Chebyshev's inequality. However, now we have that the sample size depends on ε and α as $n = -\log(\alpha/2)/2\varepsilon^2$. Hence, n grows like $-\log(\alpha)$ as $\alpha \to 0$.

2.2 Integration

Above, we noticed that computing the volume of some region $\mathcal{D} \subset \mathcal{C}$ can be written as an integral of some function $\phi(x)$ that is 1 for $x \in \mathcal{D}$ and 0 for $x \notin \mathcal{D}$. That is,

$$\mathrm{Vol}(\mathcal{D}) = \int_{\mathcal{C}} \phi(x)dx \approx \frac{1}{n}\sum_{i=1}^{n} \phi(x_i)$$

where the $x_1, \ldots, x_n$ are uniformly random points in the unit cube $\mathcal{C}$.

Now, we will do the same thing, but for an integrable function $g : \mathbb{R} \to \mathbb{R}$. That is, assume that the domain of g is $[0,1]$. Then, we can pick $x_1, \ldots, x_n \in [0,1]$ and estimate

$$\int_0^1 g(x)dx \approx \frac{1}{n}\sum_{i=1}^{n} g(x_i).$$

First, we have to determine how to pick the points x_i.

2.2.1 Deterministic Approaches

There are tons of approaches to numerical integration in existence. We will briefly discuss non-Monte Carlo methods before resuming the MC discussion. Methods referred to a "Quadrature" typically involve picking points x_i and weights ω_i and computing

$$\int_0^1 g(x)dx \approx \sum_{i=1}^n \omega_i g(x_i).$$

Let $0 = x_0 < x_1 < \ldots < x_n = 1$. The rectangular or midpoint rule yields

$$\int_0^1 g(x)dx \approx \sum_{i=1}^n (x_i - x_{i-1}) g\left(\frac{x_i + x_{i-1}}{2}\right),$$

which is just the width times the height of n rectangles. The trapezoidal rule yields

$$\int_0^1 g(x)dx \approx \sum_{i=1}^n (x_i - x_{i-1}) \left(\frac{g(x_i) + g(x_{i-1})}{2}\right),$$

which linearly interpolates between $(x_{i-1}, g(x_{i-1}))$ and $(x_i, g(x_i))$. Of course, many more sophisticated methods exist.

2.2.2 Monte Carlo Integration

In contrast to deterministic approaches, we will focus on Monte Carlo methods for numerically computing the valtue of an integral. Considering the same problem as above, we want to estimate

$$\mu = \int_0^1 g(x)dx \approx \frac{1}{n}\sum_{i=1}^n g(x_i) = \hat{\mu}.$$

If we take each x_i to be a uniform random variate $X_i \overset{\text{iid}}{\sim} \text{Uniform}\,[0,1]$, then we have for each $i = 1, \ldots, n$ that

$$\text{E}g(X_i) = \int_0^1 g(x)dx$$

and thus

$$\text{E}\left(\frac{1}{n}\sum_{i=1}^n g(X_i)\right) = \int_0^1 g(x)dx,$$

which means we have an unbiased estimator of the integral.

Furthermore, we can compute the variance and get

$$\text{Var}\left(\frac{1}{n}\sum_{i=1}^n g(X_i)\right) = \frac{1}{n^2}\sum_{i=1}^n \text{Var}\left(g(X_i)\right) = \frac{1}{n}\text{Var}\left(g(X)\right).$$

For $X \sim \text{Uniform}\,[0,1]$,

$$\sigma^2 = \text{Var}\left(g(X)\right) = \int_0^1 g(x)^2 dx - \left(\int_0^1 g(x) dx\right)^2 = \int_0^1 g(x)^2 dx - \mu^2.$$

Therefore, we have the usual conclusion that $\text{Var}\left(\frac{1}{n}\sum_{i=1}^n g(X_i)\right) = \sigma^2/n$. This can be estimated by the unbiased estimator

$$\hat{\sigma}_n^2 = \frac{1}{n-1}\sum_{i=1}^n \left[g(x_i)^2 - \left(\frac{1}{n}\sum_{j=1}^n g(x_j)\right)^2\right] = \frac{1}{n-1}\sum_{i=1}^n \left[g(x_i)^2 - \hat{\mu}_n^2\right]$$

Confidence Intervals

If we have that $\int_0^1 g(x)^4 dx < \infty$,[2] then the central limit theorem tells us that

$$\frac{\hat{\mu}_n - \mu}{\sqrt{\hat{\sigma}_n^2/n}} \xrightarrow{\text{d}} \mathcal{N}\left(0,1\right).$$

Therefore, we can use z-scores to construct an asymptotic $(1-\alpha)$-confidence interval for μ being

$$|\hat{\mu}_n - \mu| \leq z_{1-\alpha/2}\sqrt{\hat{\sigma}_n^2/n}.$$

Note that in this formulation that $\hat{\mu}_n$ and $\hat{\sigma}_n^2$ is computed from the same data. If one has the CPU cycles to spare, it is preferable to generate independent sets of data to estimate $\hat{\mu}_n$ and $\hat{\sigma}_n^2$ independently.

Steaming Computation

For a highly accurate estimate of μ, we may require a very large n. In the above formulation, computing the variance $\hat{\sigma}_n^2$ requires computation of $\hat{\mu}_n$ first. Thus, we would have to save n evaluations $g(x_i)$ in memory, which could become cumbersome. Instead, we can reformulate the above computations in a streaming fashion that does not require saving all of the evaluations to memory.

A first attempt would be to write

$$\hat{\sigma}_n^2 = \frac{1}{n-1}\sum_{i=1}^n g(x_i)^2 - \frac{1}{n(n-1)}\left(\sum_{i=1}^n g(x_i)\right)^2.$$

Hence, we could save $\sum_{i=1}^n g(x_i)$ and $\sum_{i=1}^n g(x_i)^2$ only. However, the use of subtraction could result in loss of numerical precision.

[2] i.e. finite fourth moment.

Algorithm 8 Integral via uniform Monte Carlo

Input:

function $g(x)$ on the domain $[0, 1]$.

$n \in \mathbb{N}$, the number random points to simulate.

Algorithm:

Set $S = 0$. Integral Estimate

Set $V = 0$. Variance Estimate

for $i = 1, \ldots, n$

Generate $x \sim$ Uniform $[0, 1]$.

Compute $g(x)$

$V \leftarrow (\frac{i-1}{i-2})V + \frac{1}{i}(g(x) - S)^2$

$S \leftarrow \frac{1}{i}g(x) + (\frac{i-1}{i})S$

return estimate S

return variance V

return confidence interval $S \pm z_{1-\alpha/2}\sqrt{V/n}$

Output:

Volume estimate of region $\mathcal{D}$ along with variance and confidence interval.

A more sophisticated approach is to write

$$\begin{aligned}
(n-1)\hat{\sigma}_n^2 - (n-2)\hat{\sigma}_{n-1}^2 &= \sum_{i=1}^{n} (g(x_i) - \hat{\mu}_n)^2 - \sum_{i=1}^{n-1} (g(x_i) - \hat{\mu}_{n-1})^2 \\
&= g(x_n)^2 - n\hat{\mu}_n^2 + (n-1)\hat{\mu}_{n-1}^2 \\
&= g(x_n)^2 - \frac{1}{n}\left(\sum_{i=1}^{n} g(x_i)\right)^2 + (n-1)\hat{\mu}_{n-1}^2 \\
&= g(x_n)^2 - \frac{1}{n}\left(g(x_n) + (n-1)\hat{\mu}_{n-1}\right)^2 + (n-1)\hat{\mu}_{n-1}^2 \\
&= \left(\frac{n-1}{n}\right) g(x_n)^2 - 2\left(\frac{n-1}{n}\right) g(x_n)\hat{\mu}_{n-1} - \frac{(n-1)^2 - n(n-1)}{n}\hat{\mu}_{n-1}^2 \\
\hat{\sigma}_n^2 &= \frac{n-2}{n-1}\hat{\sigma}_{n-1}^2 + \frac{1}{n}\left[g(x_n) - \hat{\mu}_{n-1}\right]^2 .
\end{aligned}$$

Therefore, we can update

$$\hat{\mu}_n = \frac{g(x_i)}{n} + \frac{n-1}{n}\hat{\mu}_{n-1}$$

and use summations to compute σ_n^2 at each step of the algorithm.

2.3 Importance Sampling

In the previous sections, we only considered sampling uniformly from $[0, 1]$. However, it is often worth considering non-uniform distributions in order to sample more efficiently. That is, starting from the above integration problem, say we want to compute

$$\mu = \int_0^1 g(t)dt = \mathrm{E}[g(X)]$$

for $X \sim \text{Uniform}\,[0, 1]$. The estimator we had before will be denoted

$$\hat{\mu}_n = \frac{1}{n}\sum_{i=1}^{n} g(x_i)$$

for $x_1, \ldots, x_n$ drawn from the Uniform $[0, 1]$ distribution. Then, given some density function $f(t)$ which has support on $[0, 1]$ and $f(t) > 0$ for all $t \in [0, 1]$, we can write

$$\mu = \int_0^1 g(t)dt = \int_0^1 g(t)\frac{f(t)}{f(t)}dt = \int_0^1 \frac{g(t)}{f(t)}f(t)dt = \mathrm{E}[g(Z)/f(Z)]$$

for $Z \sim f(z)$. Numerically speaking, we can simulate $z_1, \ldots, z_n$ from $f(z)$ and then estimate the integral to be

$$\tilde{\mu}_n = \frac{1}{n}\sum_{i=1}^{n} \frac{g(z_i)}{f(z_i)}.$$

Then, the question to consider is, "What should $f(z)$ be in order to get the most efficient estimate?"

Based on the above equations, we can derive the variance for our new estimator to be

$$\begin{aligned}\text{Var}\,(g(Z)/f(Z)) &= \mathrm{E}\left[\left(\frac{g(Z)}{f(Z)} - \mu\right)^2\right] \\ &= \mathrm{E}\left[\frac{g(Z)^2}{f(Z)^2}\right] - \mu^2 = \int_0^1 \frac{g(t)^2}{f(t)}dt - \mu^2.\end{aligned}$$

In contrast, the variance of $g(X)$ for X uniform is $\int_0^1 g(t)^2dt - \mu^2$. Thus, we can consider the difference

$$\text{Var}\,(g(X)) - \text{Var}\left(\frac{g(Z)}{f(Z)}\right) = \int_0^1 g(t)^2\left(1 - \frac{1}{f(t)}\right)dt.$$

Thus, we want this integral to be as big as possible to indicate a large drop in the variance. Solving for the perfect f is typically not doable. However, the above equation gives us valuable intuition. In particular, if $g(t)^2$ is *big* then we want $f(t) > 1$ and if $g(t)^2$ is *small* then we want $f(t) < 1$. Thus we are sampling more or less frequently than the uniform distribution depending on what f is.

Example 2.3.1. *If we want to numerically estimate the integral of* $g(t) = t^2 + t^3$*, we can try simulating from the uniform distribution—i.e.* $\text{Beta}(1,1)$*—and the* $\text{Beta}(2,1)$ *distribution with pdf* $f(t) = 2t$*. Computing the variance difference from above yields*

$$\begin{aligned}\int_0^1 (t^2+t^3)^2\left(1-\frac{1}{2t}\right)dt &= \int_0^1 (t^4+2t^5+t^6)dt - \frac{1}{2}\int_0^1 (t^3+2t^4+t^5)dt \\ &= \left(\frac{1}{5}+\frac{1}{3}+\frac{1}{7}\right) - \left(\frac{1}{8}+\frac{1}{5}+\frac{1}{12}\right) = \frac{15}{56} \approx 0.268.\end{aligned}$$

The variance under the uniform distribution is

$$\int_0^1 (t^2+t^3)^2 dt - \left[\int_0^1 (t^2+t^3)\right]^2 \approx 0.3359.$$

Hence, dropping by 0.268 is a big drop in the variance of this estimator.

2.4 Stratified Sampling

Another useful tool for variance reduction is to take the region you are trying to integrate over, break it into k pieces (or strata), and sample from each to estimate the integral.

To integrate $g(x)$ over the interval $[0,1]$, we partition the interval into disjoint sets $A_1, \ldots, A_k$ such that $[0,1] = \bigcup_{i=1}^k A_i$. Then, we can write

$$\mu = \int_0^1 g(x)dx = \sum_{i=1}^k \int_{A_i} g(x)dx = \sum_{i=1}^k \omega_i \int_{A_i} g(x)\frac{dx}{\omega_i} = \sum_{i=1}^k \omega_i \mathrm{E}_{A_i}[g(x)]$$

where $\omega_i = |A_i| = \int_{A_i} g(x)dx$. The point of weighting with ω_i is so that each integral becomes an expectation. Note that $\omega_1 + \ldots + \omega_k = 1$. We can also change the probability distribution for sampling on each A_i if we want to complicate this further.

For each A_i, we sample n_i points from A_i and denote these points as $x_{i,j}$. Then, our estimator becomes

$$\hat{\mu}_n^{\text{ss}} = \sum_{i=1}^k \frac{\omega_i}{n_i} \sum_{j=1}^{n_i} g(x_{i,j})$$

and the variance of the estimator becomes

$$\text{Var}\left(\hat{\mu}_n^{\text{ss}}\right) = \sum_{i=1}^k \frac{\omega_i^2}{n_i} \text{var}_{A_i}(g(X))$$

where $\text{var}_{A_i}(g(X))$ is the variance over each stratum A_i. Therefore, our goal is to chose the A_i and n_i to reduce the variance.

One way to get a variance reduction is to choose each $n_i = n\omega_i$, which simply says to sample from each A_i proportional to its size. That is,

$$\begin{aligned}
\mathrm{Var}\left(\hat{\mu}_n^{\mathrm{ss}}\right) &= \sum_{i=1}^{k} \frac{\omega_i}{n} \mathrm{var}_{A_i}(g(X)) \\
&= \frac{1}{n} \sum_{i=1}^{k} \omega_i \mathrm{E}_{A_i}\left[(g(X) - \mathrm{E}_{A_i}[g(X)])^2\right] \\
&= \frac{1}{n} \sum_{i=1}^{k} \omega_i \mathrm{E}_{A_i}\left[(g(X) - \mathrm{E}[g(x)] + \mathrm{E}_{A_i}[g(X)] - \mathrm{E}[g(x)])^2\right] \\
&= \frac{1}{n} \sum_{i=1}^{k} \omega_i \left(\mathrm{E}_{A_i}\left[(g(X) - \mathrm{E}[g(x)])^2\right] - (\mathrm{E}_{A_i}[g(X)] - \mathrm{E}[g(x)])^2\right) \\
&= \mathrm{Var}\left(\hat{\mu}_n\right) - \frac{1}{n} \sum_{i=1}^{k} \omega_i \left(\mathrm{E}_{A_i}[g(X)] - \mathrm{E}[g(x)]\right)^2,
\end{aligned}$$

which is less than or equal to $\mathrm{Var}\left(\hat{\mu}_n\right)$. The main intuition here is that we are forcing the sampling to be more evenly spread out. However, this approach can still be outperformed with more clever thought. In particular, this above partitioning does not take the function g into account at all.

Example 2.4.1. *If we simply want to integrate $g(x) = x$ on $[0, 1]$, we generate $x_1, \ldots, x_n$ uniformly and get*[3]

$$\hat{\mu}_n = \frac{1}{n} \sum_{i=1}^{n} x_i \quad \text{and} \quad \mathrm{Var}\left(\hat{\mu}_n\right) = \frac{1}{12n}.$$

If we instead partition into intervals $[0, 0.5]$ and $[0.5, 1]$ with $n/2$ points in each, we get

$$\hat{\mu}_n^{\mathrm{ss}} = \sum_{i=1}^{2} \frac{1/2}{n/2} \sum_{j=1}^{n/2} x_{i,j} = \frac{1}{n} \sum_{i=1}^{2} \sum_{j=1}^{n/2} x_{i,j},$$

which is the same as $\hat{\mu}_n$ except that exactly half of the samples are in each of the two subintervals. The variance of this estimator is

$$\mathrm{Var}\left(\hat{\mu}_n^{\mathrm{ss}}\right) = \sum_{i=1}^{2} \frac{(1/2)^2}{n/2} \mathrm{var}_{A_i}(X) = \frac{1}{2n} \sum_{i=1}^{2} \frac{(1/2)^2}{12} = \frac{1}{48n}.$$

Hence, our variance has reduced by a factor of 4.

[3] Recall that the variance of the Uniform $[a, b]$ distribution is $(b - a)^2/12$.

Chapter 3

Stochastic Optimization

Introduction

In the last chapter, we saw how Monte Carlo methods can be used to estimate volumes and integrals. In this chapter we will consider a different but also important application of Monte Carlo methods: Optimization.

In many areas of application, we find ourselves having to solve similar minimization / maximization problems. For example, in parametric statistics, we often wish to find a model that maximizes the likelihood. Alternatively, we may want to choose model parameters that best minimize a loss or risk function. In this chapter, we will formulate all problems as minimizing some loss function $L(\theta)$ which maps a vector $\theta \in \mathbb{R}^d$ into $\mathbb{R}$. The goal will be to find $\theta^* = \arg\min_\theta\{L(\theta)\}$. Of course, any of the ideas below can be easily reformulated into a maximization problem.

There are also two settings we will consider: noise-free and noisy optimization. In the first case, we simply try to minimize the deterministic function $L(\theta)$, which nevertheless may be a very complex function to minimize. In the second case, we do not get to observe $L(\theta)$, but instead observe $L(\theta)+\varepsilon_\theta$ where ε_θ is a random error in the measurement that may depend on θ.

3.1 Basic Stochastic Search

The ideas in this section come from Chapter 2 of Spall (2005). These methods are all relatively simple methods to code and implement and, though simple, can still be applicable to real world problems.

3.1.1 Simple Random Search

The first and by far easiest method of stochastic search and optimization is to simply pick points (θ's) uniformly at random. This is outlined in Algorithm 9. In practice, we don't have to simulate θ's uniformly at random, but can use other distributions

Algorithm 9 Simple Random Search

Input:
Loss function $L(\theta)$ to minimize.
Domain Θ to search over.
Maximum number of samples n.
Algorithm:
Pick $\theta_{\text{best}} \in \Theta$ uniformly at random
Set $L_{\text{best}} = L(\theta_{\text{best}})$
for i from 2 to n
Pick $\theta \in \Theta$ uniformly at random
Compute $L = L(\theta)$
if $L < L_{\text{best}}$ (i.e. if we do improve)
Set $L_{\text{best}} = L$ and $\theta_{\text{best}} = \theta$
Output:
$\{\theta_{\text{best}}, L_{\text{best}}\}$

as well. However, with no additional information on $L(\theta)$, sampling uniformly is best.

Under some mild conditions on the loss function L, it can be shown that Algorithm 9 will find the optima as $n \to \infty$. The theorem presented in Spall (2005) notes that the optimum can be found when

- The minimizer θ^* is unique, and for all $\varepsilon > 0$, $\inf_{\|\theta - \theta^*\| > \varepsilon} L(\theta) > L(\theta^*)$
- The minimum value $L(\theta^*) > -\infty$.
- A "continuity"-like condtion: for all $\varepsilon > 0$, there exists $\delta > 0$ such that

$$\mathrm{P}\left(L(\theta) < L(\theta^*) + \varepsilon\right) \geq \delta,$$

  which basically means that θ^* can be some isolated singleton point with respect to how we are randomly generating θ's.

However, we won't prove that in these notes as we would have to discuss what *convergence almost surely* means first as well as other analysis topics.

3.1.2 Localized Random Search

While the simple search method from the previous section is valid and will suceed as $n \to \infty$, it doesn't make use of the loss function $L(\theta)$ at all. Typically, we would expect some smoothness to the function L. That is, if we find a "small" value at θ_0, then we would expect θ's close to θ_0 to also give small values of L.

Algorithm 10 Localized Random Search

Input:
Loss function $L(\theta)$ to minimize.
Domain Θ to search over.
Maximum number of samples n.
Distribution f to sample from (typically standard normal)
Algorithm:
Pick $\theta_1 \in \Theta$ uniformly at random
Set $L_1 = L(\theta_1)$
for i from 2 to n
Pick v from distribution f
Set $\theta_i = \theta_{i-1} + v$
if $\theta_i \notin \Theta$
Set $\theta_i \leftarrow \theta_{i-1}$ and $L_i \leftarrow L_{i-1}$
Go to next i
Compute $L_i = L(\theta_i)$
if $L_i \geq L_{i-1}$ (i.e. if we don't improve)
Set $L_i \leftarrow L_{i-1}$ and $\theta_i \leftarrow \theta_{i-1}$
Output:
$\{\theta_n, L_n\}$

In the localized search method detailed in Algorithm 10, we move from one θ to the next by randomly perturbing it by some random vector v. That is, the new θ becomes $\theta + v$. Thus, this method randomly walks around the domain Θ looking for the minimum. It can be shown that if

- L is continuous
- Θ is bounded
- $v \sim \mathcal{N}(0, cI_d)$ for some choice of $c > 0$

then our sequence of θ's will get arbitrarily close to the minimizer θ^* as $n \to \infty$. If there is more than one minimizer—i.e. θ^* is not unique—then this algorithm will eventually become close to one of the minmizers.

Remark 3.1.1 (Boundary Conditions)**.** *Note that if Θ is a bounded subset of $\mathbb{R}^d$ and if the v's come from the normal distribution then we could pick a $\theta \notin \Theta$. In this case, we would need to have our algorithm check to make sure we haven't left the domain of interest.*

3.1.3 Enhanced Localized Search

The last algorithm we consider in this section is almost identical to the above localized search method. However, it is "enhanced" by adding a bias term that pushes the random walk in a given direction if that direction looks to be minimizing the loss function L. This bias term gives the random walk some inertial to keep drifting in the same direction (roughly). This is detailed in Algorithm 11.

Specifically, the update step from the localized search,

$$\text{LRS:} \quad \theta_i \leftarrow \theta_{i-1} + v,$$

is replaced with

$$\text{ERS:} \quad \theta_i \leftarrow \theta_{i-1} + b_{i-1} + v,$$

where b_{i-1} is the bias term computed on the previous step of the search algorithm. If the new θ_i gives an improvement in the loss value—i.e. $L(\theta_i) < L(\theta_{i-1})$—then the new bias term is

$$b_i \leftarrow 0.2b_{i-1} + 0.4v.$$

Otherwise, the new bias term is a shrunken version of the previous term $b_i \leftarrow 0.5b_{i-1}$. The choices of these scaling values (0.2,0.4,0.5) were picked by the algorithm designers, but other values could be considered as well. There is nothing inherently perfect about those choices other than that they seem to work well in practice. More references to such enhanced localized search methods as discussed on page 45, chapter 2, of Spall (2005).

3.1.4 Noisy Loss Function

The above algorithms were stated assuming that the loss function L is noise-free—i.e. deterministic. Trying to minimize a deterministic function in multiple dimensions can be a hard task. Adding noise into it can make it even more arduous. Noise or measurement errors can enter into a loss evaluation in many ways. In these notes, we will just consider additive noise. That is, we don't observe $L(\theta)$, but instead observe

$$L(\theta) + \varepsilon_\theta$$

where ε_θ represents some additive noise that can vary depending on the value of θ. Typically, we will just consider the simplest setting of having $\varepsilon_\theta \overset{\text{iid}}{\sim} \mathcal{N}\left(0, \eta^2\right)$ for some unknown $\eta^2 > 0$. We will consider two approaches to handling noisy measurements: multiple evaluations and thresholding.

Based on the usual ideas of unbiased estimators and the law of large numbers, we can estimate the true value of $L(\theta)$, by making k evaluations $L(\theta) + \varepsilon_\theta$ and averaging them together. In which case, we would have measurements $L_1, \ldots, L_k$ and an average value of $\bar{L}$. As a result,

$$\text{E}\bar{L} = L(\theta) \quad \text{and} \quad \text{Var}\left(\bar{L}\right) = \eta^2/k.$$

Algorithm 11 Enhanced Localized Search

Input:
Loss function $L(\theta)$ to minimize.
Domain Θ to search over.
Maximum number of samples n.
Distribution f to sample from (typically standard normal)

Algorithm:
Pick $\theta_1 \in \Theta$ uniformly at random
Set $L_1 = L(\theta_1)$
Set $b_1 = 0$
for i from 2 to n
  Pick v from distribution f
  Set $\theta_i = \theta_{i-1} + b_{i-1} + v$ (try adding v)
  if $\theta_i \in \Theta$ **and** $L_i := L(\theta_i) < L(\theta_{i-1})$
    Set $b_i = 0.2b_{i-1} + 0.4v$
    Go to next i
  Set $\theta_i = \theta_{i-1} + b_{i-1} - v$ (try subtracting v)
  if $\theta_i \in \Theta$ **and** $L_i := L(\theta_i) < L(\theta_{i-1})$
    Set $b_i = 0.2b_{i-1} - 0.4v$
    Go to next i
  Set $b_i = 0.5b_{i-1}$
  Set $L_i \leftarrow L_{i-1}$ and $\theta_i \leftarrow \theta_{i-1}$ (or keep θ unchanged)

Output:
$\{\theta_n, L_n\}$

Thus, we have reduced the variance by a factor of $1/k$. However, this would require k-times more evaluations of L at every step. This may be prohibitively expensive depending on how hard it is to compute loss measurements.

Instead, we can set a threshold for improvement. That is, in the originally stated (noise-free) algorithms, we update θ_{i-1} to θ_i if $L(\theta_i) < L(\theta_{i-1})$. In this noisy-setting, we can pick a threshold τ_i, which could be dependent on the iteration i, and update the parameter vector θ if

$$L(\theta_i) + \varepsilon_i < L(\theta_{i-1}) + \varepsilon_{i-1} - \tau_i.$$

The intuition comes from hypothesis testing. If τ_i is, say, two standard deviations, then if $L(\theta_i) > L(\theta_{i-1}$ then

$$\mathrm{P}\left(L(\theta_i) + \varepsilon_i < L(\theta_{i-1}) + \varepsilon_{i-1} - \tau_i\right) \leq 2.3\%.$$

This means that we would have only a 2.3% chance of picking a worse choice of parameters θ. Of course, this conceptualization assumes we know what the unknown

noise η^2 is. In practice, we would have to treat τ as a tuning parameter. If τ is small, we jump around Θ more liberally. If τ is large, we require a large drop in the observed loss value before we jump to a new θ.

3.2 Stochastic Approximation

In this final section, we will consider some methods of stochastic approximation, which are either concerned with root finding or optimization. Most methods can be used for either problem. Mainly, if we want to minimize a loss function $L(\theta)$, we are effectively looking for the root of

$$\frac{d}{d\theta}L(\theta) = 0.$$

Of course, this assumes we have access to the derivative $\frac{dL}{d\theta}$. We also assume in these final sections that all measurements are noisy.

3.2.1 Gradient Methods

Earlier in these notes, we discussed both the Newton-Raphson and bisection algorithm as techniques for root finding for deterministic functions. However, these methods will not work well in the presence to noise. To solve the noisy root-finding problem (i.e. finding a critical point of the loss function), we will consider the Robbins-Monro algorithm

In this section, we will assume that the loss function $L(\theta) = \mathrm{E}[Q(\theta, v)]$ where Q is the noisy loss function with some noise variable v. Then, let

$$y(\theta) = \frac{\partial Q(\theta, v)}{\partial \theta}$$

be the function for which we want to find a root. The function $y(\theta)$ is a noisy measurement of the gradient of Q.

Robbins-Monro

The Robbins-Monro algorithm dates back to the 1950's and will look very similar to Newton-Raphson. In short, we require a sequence of step-sizes or sometimes called a *gain* sequence a_k such that

$$\sum_{k=1}^{\infty} a_k = \infty \quad \text{and} \quad \sum_{k=1}^{\infty} a_k^2 < \infty,$$

which basically means that the step-sizes $a_k \to 0$ but not too fast. This allows us to explore the space Θ, but eventually calm down into a final solution.

Algorithm 12 The Robbins-Monro Algorithm

Input:

$y : \Theta \to \mathbb{R}$,	a function to find the root
$\theta_0 \in \Theta$,	an initial value in Θ
$\tau \in \mathbb{R}^+$,	a threshold for convergence

Algorithm:

For $k \geq 1$
  $\theta_k = \theta_{k-1} - a_k y(\theta_{k-1})$
  if $\|\theta_k - \theta_{k-1}\| < \tau$
    stop
return θ_k

Output:

θ_k such that $y(\theta_k) \approx 0$

Given parameters from the previous set θ_k, the update step for the Robbins-Monro algorithm is

$$\theta_{k+1} \leftarrow \theta_k - a_k y(\theta_k).$$

The algorithm is detailed in Algorithm 12. A standard choice for the step-size sequence a_k is $a_k = a/k$ for some constant $a > 0$. Besides this sequence, a few more technical conditions must be statisfied to prove convergence. These include additive mean-zero noise, a gaurenttee that the variance doesn't blow up, and convexity for $L(\theta)$. However, convergence criteria take a lot of background knowledge to discuss properly and thus will be skipped in these notes.

3.2.2 Gradient-free Methods

Very often, we do not have access to the dervivative of a noise loss function. That is for some L and Q such that $L(\theta) = \mathrm{E}[Q(\theta, v)]$, we can evalutate Q at chosen values of θ but cannot evaluate $\partial Q/\partial\theta$. This problem necessitates gradient-free methods. These methods typically revolve around sampling points to estimate the gradient. Most noteably, we will discuss the Kiefer–Wolfowitz method, whic is also known as the finite difference method. A second method to consider is the *simultaneous perturbation* method, which tries to reduce the number of sample points to use to estimate the gradient.

Kiefer–Wolfowitz

Much like the Robbins-Monro algorithm from the previous section, the update step for the Kiefer-Wolfowitz algorithm is

$$\theta_{k+1} \leftarrow \theta_k - a_k \hat{y}_k(\theta_k)$$

Algorithm 13 The Kiefer–Wolfowitz Algorithm

Input:

$y : \Theta \to \mathbb{R}$,	a function to find the root
$\theta_0 \in \Theta$,	an initial value in Θ
$\tau \in \mathbb{R}^+$,	a threshold for convergence

Algorithm:

For $k \geq 1$
  Estimate the gradient $\hat{y}_k(\theta_{k-1})$
  $\theta_k = \theta_{k-1} - a_k \hat{y}_k(\theta_{k-1})$
  if $\|\theta_k - \theta_{k-1}\| < \tau$
   stop
return θ_k

Output:

θ_k such that $y(\theta_k) \approx 0$

where $\hat{y}_k(\theta_k)$ is an estimate of the gradient at the input θ_k. To estimate the gradient, we require a second step-size sequence c_k. Then, we get

$$\hat{y}_k(\theta_k) = \begin{pmatrix} \frac{1}{2c_k}\left[Q(\theta_k + c_k e_1) - Q(\theta_k - c_k e_1)\right] \\ \vdots \\ \frac{1}{2c_k}\left[Q(\theta_k + c_k e_d) - Q(\theta_k - c_k e_d)\right] \end{pmatrix}$$

where $e_i = (0, \ldots, 0, 1, 0, \ldots, 0)$ are the standard basic vectors in $\mathbb{R}^d$ with a 1 in the ith position. Thus, at each iterate, we need to evaluate the loss function at $2d$ different inputs. This algorithm is detailed in Algorithm 13.

The convergence of this method depends on proper selection of the step-size or gain sequences a_k and c_k. We require:

- positivity: $a_k > 0$ and $c_k > 0$ for all k
- convergence to zero: $a_k \to 0$ and $c_k \to 0$
- summations: $\sum_{k=0}^{\infty} a_k = \infty$, $\sum_{k=0}^{\infty} a_k c_k < \infty$, $\sum_{k=0}^{\infty} a_k^2 / c_k^2 < \infty$.

One natural choice for these sequences is $a_k = a/k$ for some $a > 0$ as before and $c_k = c/k^\alpha$ for some $c > 0$ and some $\alpha \in (0, 1/2)$, e.g. $\alpha = 1/3$ or $\alpha = 1/4$, but $\alpha \neq 1/2$ as then the final sum will not converge.

Random Perturbation FDSA

This method is the same as Kiefer-Wolfowitz except for how the gradient is esimated. Instead of picking $2d$ points along the Cartesian axes, we randomly draw a

perturbation vector v from some suitable distribution (e.g. normal) and estimate the gradient at only 2 points. The gradient equation is

$$\hat{y}_k(\theta_k) = \frac{[Q(\theta_k + c_k v) - Q(\theta_k - c_k v)]}{2c_k} \begin{pmatrix} v_1 \\ \vdots \\ v_d \end{pmatrix}.$$

Thus, a method like this is nicer in high dimensions especially if Q is costly to evaluate.

Simultaneous Perturbation FDSA

A similar idea is the simultaneous perturbation method. In this case, we choose a different perturbation vector v with entries of ± 1 with probability $1/2$ each (this is known as the Rademacher distribution). Other distributions can also be considered, but this setup is closest to the classic Kiefer-Wolfowitz algorithm. The gradient estimation equation is

$$\hat{y}_k(\theta_k) = \frac{[Q(\theta_k + c_k v) - Q(\theta_k - c_k v)]}{2c_k} \begin{pmatrix} 1/v_1 \\ \vdots \\ 1/v_d \end{pmatrix}.$$

Note that for Rademacher vectors v, this approach is the same idea as the random perturbation method as $1/v_i = v_i$ when $v_i = \pm 1$. However, the point is that other distributions can also be chosen as long as v doesn't get close to zero with high probability.

3.3 Example: Least Squares and Neural Networks

If we have a collection of input and output pairs, (x_k, z_k), a common desire is to fit a function that best approximates the relation between x_k and z_k. That is, we have a parametric function $h(\theta, x_k)$ and errors $\varepsilon_k = z_k - h(\theta, x_k)$. The goal is to pick a parameter vector θ^* to minimize the expected value of $\sum_{k=1}^n \varepsilon_k^2$. That is,

$$\text{minimize:} \quad L(\theta) = \frac{1}{2n} \sum_{k=1}^{n} \mathrm{E}\left(z_k - h(\theta, x_k)\right)^2.$$

Taking a derivative means we can consider the Robbins-Monro algorithm with the following update equation:

$$\theta_{k+1} = \theta_k - a_k \left[z_k - h(\theta, x_k)\right] \frac{\partial h}{\partial \theta}(\theta_k, x_k).$$

In this case, we consider each input-output pair one at a time rather than all at once. This is useful, for example, with large and/or streaming data sets.

This above formulation can be applied to any least squares problem where h and the gradient of h are able to be computed. For example, h could be some nonlinear regression equation or a function described by a neural network. The segmented construction of a neural network allows for evaluating the gradient of h.

Appendix A

Appendix

A.1 Change of Variables

A tool that is integral to many of the theorems and proofs in these notes is the change-of-variables formula. Imagine we are in $\mathbb{R}^n$ with coordinate variables $(x_1, \ldots, x_n)$. But now we wish to consider a new coordinate system $(y_1, \ldots, y_n)$. This allows us to define the $n \times n$ Jacobian matrix

$$J = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_n}{\partial x_1} & \cdots & \frac{\partial y_n}{\partial x_n} \end{pmatrix}.$$

The determinant of J will be of particular interest and is often also referred to simply as the *Jacobian*. Note that we want this determinant to be non-zero. Otherwise, the mapping from $\boldsymbol{x} \to \boldsymbol{y}$ is not invertible.[1] The change-of-variables formula is as follows. Note that more general formulations of this result exist, but we stick with a simpler one suitable for our purposes.

Proposition A.1.1 (Change of Variables)**.** *Let $U \subset \mathbb{R}^n$ be an open set, and let the change of variables function $\varphi : \boldsymbol{x} \to \boldsymbol{y}$ be one-to-one, differentiable, have continuous partial derivatives, and non-zero Jacobian for all $\boldsymbol{x} \in U$. Then, for any real-valued function f that is continuous with support contained in $\varphi(U)$,*

$$\int_{\varphi(U)} f(\boldsymbol{y}) d\boldsymbol{y} = \int_U f(\boldsymbol{x}) |\det(J)| d\boldsymbol{x}.$$

Example A.1.2 (Spherical Coordinates)**.** *An archetypal application is switching from Cartesian to polar coordinates. In $\mathbb{R}^3$, we can define two coordinate systems (x, y, z) and (r, θ, ϕ) such that*

$$x = r \sin\phi \cos\theta, \quad y = r \sin\phi \sin\theta, \quad z = r \cos\phi.$$

[1] see the Inverse Function Theorem.

In this case, the Jacobian matrix is

$$J = \begin{pmatrix} \frac{\partial x}{\partial r} & \frac{\partial x}{\partial \phi} & \frac{\partial x}{\partial \theta} \\ \frac{\partial y}{\partial r} & \frac{\partial y}{\partial \phi} & \frac{\partial y}{\partial \theta} \\ \frac{\partial z}{\partial r} & \frac{\partial z}{\partial \phi} & \frac{\partial z}{\partial \theta} \end{pmatrix} = \begin{pmatrix} \sin\phi\cos\theta & r\cos\phi\cos\theta & -r\sin\phi\sin\theta \\ \sin\phi\sin\theta & r\cos\phi\sin\theta & r\sin\phi\cos\theta \\ \cos\phi & -r\sin\phi & 0 \end{pmatrix},$$

and the determinant is $\det(J) = r^2\sin\phi$. *Let* $\varphi : (r,\phi,\theta) \to (x,y,z)$. *If we have a region* $U \subset \mathbb{R}^3$ *and function* f, *then*

$$\iiint_{\varphi(U)} f(x,y,z)dx\,dy\,dz = \iiint_U f(r\sin\phi\cos\theta, r\sin\phi\sin\theta, r\cos\phi)r^2\sin\phi\,dr\,d\phi\,d\theta.$$

The change-of-variables formula is critical to working with probabilities and random variables. This is because probabilities are actually measures which are actually integrals. More precisely, consider real valued random variables X, Y with joint density function $f_{X,Y}(x,y)$. Then,

$$\mathrm{P}\left((X,Y) \in U\right) = \iint_U f_{X,Y}(x,y)dx\,dy.$$

If we wish to transform into new random variables W, Z where $(W,Z) = \varphi(X,Y)$, then

$$\mathrm{P}\left((W,Z) \in \varphi(U)\right) = \iint_{\varphi(U)} f_{W,Z}(w,z)dw\,dz = \iint_U f_{X,Y}(x(w,z),y(w,z))\left|\frac{\partial x}{\partial w}\frac{\partial y}{\partial z} - \frac{\partial x}{\partial z}\frac{\partial y}{\partial w}\right|dw\,dz.$$

This implies that the joint density of W and Z is

$$f_{W,Z}(w,z) = f_{X,Y}(x(w,z),y(w,z))\left|\frac{\partial x}{\partial w}\frac{\partial y}{\partial z} - \frac{\partial x}{\partial z}\frac{\partial y}{\partial w}\right|,$$

which is used (among other places) in the proof of the Box-Muller transform and the Ratio of Uniforms method.

Pro Tip : $\det(AB) = \det(A)\det(B)$ and $\det(A^{-1}) = \det(A)^{-1}$.

Bibliography

Russell CH Cheng and GM Feast. Gamma variate generators with increased shape parameter range. *Communications of the ACM*, 23(7):389–395, 1980.

George Fishman. *Monte Carlo: concepts, algorithms, and applications*. Springer Science & Business Media, 2013.

Albert J Kinderman and John F Monahan. Computer generation of random variables using the ratio of uniform deviates. *ACM Transactions on Mathematical Software (TOMS)*, 3(3):257–260, 1977.

William H Press, Saul A Teukolsky, William T Vetterling, and Brian P Flannery. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

James C Spall. *Introduction to stochastic search and optimization: estimation, simulation, and control*, volume 65. John Wiley & Sons, 2005.