Math 459: Lecture 14

Todd Kuffner

Last time

Determining when an elementary function has an elementary antiderivative is not easy, but at least now you know why.

Monte Carlo integration is a method for approximating integrals (and hence approximate Bayesian inference).

Since Monte Carlo methods utilize random sampling, we require a source of randomness.

Random Number Generators

Question: How do we generate random numbers from some set or interval?

More convenient question: how to generate a stream of independent $\text{Unif}(0, 1)$ variables

- we will see that if we can do this, then we can generate random numbers more generally

Truly random numbers from physical generators

Truly random number generators from physical processes:

- coin tosses, dice, etc.
- build a device that delivers a random number from the outcome of some physical process (e.g. radioactive particle emission) which is believed to be truly random

Some disadvantages:

1. cannot re-run a simulation after changing some parameters or encountering an error; a physical random number generator cannot be restarted, so we would have to store the entire sequence of numbers (lots of storage)
2. usually not fast enough

Pseudo-random number generators

A computational alternative to physical random number generators is pseudo-random number generators.

- random numbers are **simulated** using an *algorithm*
- such numbers are not truly random; they are *pseudo-random*
- 'pseudo-random' means the sequence behaves *as if* it were random, in that it can 'fool' an arsenal of tests for randomness
- the idea is to generate a sequence of random numbers $x_1, x_2, \ldots$ and convert them to $u_i \in [0,1]$ and hopefully have that $u_1, u_2, \ldots$ are approximately uniformly distributed

Since pseudo-random number generators dominate statistical practice, we just say RNG and 'pseudo' is understood to be implied.

John von Neumann: *Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin.*

(perhaps explains his dying request...)

Terminology

seed the initial value of the sequence; this determines the sequence of numbers–by choosing the same seed one can repeat the simulation and get the same values

state a variable denoting the current position in the finite vector of possible states

`rand` the RNG specifies a function, `rand`, that takes the current state and updates to a new state

period suppose we repeatedly call the function `rand`, producing x_i for $i \geq 1$, and suppose that the state of the generator when x_{i_0} is produced is the *same* as when $x_{i_0 - P}$ was produced
$\Rightarrow$ then $x_i = x_{i-P}$ holds for all $i \geq i_0$, i.e. from i_0 onwards, the generator is a **deterministic cycle with period** P

Some RNGs cycle at different periods depending on the initial state, but let's ignore that and suppose an RNG has a fixed period of P s.t. $x_{i+P} = x_i \; \forall \; i$.

Period Length

Obviously, small P means the sequence will not be very random.

Linear congruential operators sometimes have $P = 2^{32} - 1$, which is actually too small for Monte Carlo.

Mersenne twister (default in `R`) has $P = 2^{19937} - 1 > 10^{6000}$, which is *pretty good.*

Linear Congruential Generators

An RNG based on simple recursions utilizing *modular arithmetic* is the linear congruential generator (LCG):

$$x_i = a_0 + a_1 x_{i-1} \mod M$$

With $a_0 = 0$, this is the multiplicative congruential generator (MCG):

$$x_i = a_1 x_{i-1} \mod M.$$

An LCG must be slower than a MCG, and not really better quality. Thus MCGs are more commonly used.

Generalization of MCG is **multiple recursive generator (MRG)**

$$x_i = a_1 x_{i-1} + a_2 x_{i-2} + \cdots + a_k x_{i-k} \mod M, \;\; k \geq 1, \;\; a_k \neq 0$$

<u>All of these</u> methods produce sequences of integer values modulo M, i.e.

$x_i \in \{0, 1, \ldots, M-1\}$.

Properties

- LCG has at most M *different* values, and thus has period $P \le M$
- MCG cannot have $x_i = 0$ since then the sequence remains at 0
- MRG will start to repeat when k consecutive values $x_i, \ldots, x_{i+k-1}$ duplicate a previously-seen k-tuple of values; there are M^k of these, so the state with k consecutive 0s is not allowed
  $\Rightarrow P \le M^k - 1$

How to choose M?

Some proposals:

- a large prime number, e.g. $2^{31} - 1$; a prime number of the form $2^k - 1$ is called a Mersenne prime (there are 49 known Mersenne primes)
- $M = 2^r$ for an integer $r > 1$
- $M = 2$: an MRG with this choice of M is called a **linear feedback shift register (LFSR)** generator

What makes a RNG 'good'?

Uniformity The generated numbers $x_1, \ldots, x_P$ converted to $u_1, \ldots, u_P$ should be approximately $\mathrm{Unif}(0,1)$.

- this is not enough, since for example $x_n = x_{n-1} + 1 \mod m$ achieves perfect uniformity on values 0 through $m-1$ by going through them in order–a perfect discrete uniform distribution for the u_i—but successive *pairs* of random numbers are far from independent
- *stronger requirement:* successive pairs $(u_1, u_2), (u_2, u_3), \ldots, (u_P, u_1)$ are approximately $\mathrm{Unif}[0,1]^2$-distributed
- generalization: k-equidistributed RNGs; instead of pairs, k-tuples are s.t. the proportion of points in the k-dimension unit hypercube which lie in a particular subcube is proportion to the subcube's volume relative to the total volume

For example, $k = 623$ for the Mersenne twister (with 32 bits accuracy).

Tests many statistical tests exist for uniformity, as well as independence, predictability

How do we generate random samples?

Some Probabilistic Witchcraft: Given a random variable that is uniform on $[0, 1]$, we can transform to a random variable having any other distribution.

Definition (The Probability Integral Transform)

Consider any random variable X having continuous CDF F_X. Then the random variable $Y = F_X(X) \sim \text{Unif}(0, 1)$.

Explanation: Let $U \sim \text{Unif}(0, 1)$. The CDF of U, F_U, is $\Pr(U \leq c) = c$ for $c \in [0, 1]$. Consider the random variable $Y = F_X(X)$. The CDF of Y, F_Y satisfies

$$F_Y(t) = \Pr(Y \leq t) = \Pr(F_X(X) \leq t) = \Pr(X \leq F_X^{-1}(t)) = F_X(F_X^{-1}(t)) = t$$

which is the same as F_U on the interval $[0, 1]$, i.e. $Y = F_X(X) \sim \text{Unif}(0, 1)$.

Definition (Equivalent Statement)

For any continuous CDF F, if $U \sim \text{Unif}(0,1)$, then

$$X = F^{-1}(U) = \inf\{x : F(x) \geq U\}$$

has a CDF equal to F, i.e. $F^{-1}(U)$ defines a random variable X having CDF F.

- when F^{-1} is available for the target density, then this method is exact

Inverse Transform Sampling Method

Example: Generating Uniform on $[a, b]$

Goal: generate $X \sim \text{Unif}(a, b)$, with CDF $F_X(y) = \int_a^y (b-a)^{-1} dx = \frac{y-a}{b-a}$.

- We have $F_X^{-1}(F_X(y)) = y$, and we find $F_X^{-1}(y) = (b-a)y + a$.
- Now draw $U \sim \text{Unif}(0, 1)$.
- Then
$$X = F_X^{-1}(U) = a + (b-a)U.$$

Example: Generating Exponential

Goal: generate $X \sim \text{Exp}(\lambda)$, with CDF $F_X(y) = \int_0^y \lambda e^{-\lambda x} dx = 1 - e^{-\lambda y}$.

- We have $F_X^{-1}(y) = -\frac{\log(1-y)}{\lambda}$.
- Now draw $U \sim \text{Unif}(0, 1)$.
- Then

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

Example: Generating Normal

Goal: generate $X \sim \mathcal{N}(\mu, \sigma^2)$, with CDF

$$\Phi(y) = \frac{1}{\sqrt{2\pi\sigma^2}} \int_{-\infty}^{y} e^{-(x-\mu)^2/2\sigma^2} dx.$$

Last lecture: we learned that we can't compute this in closed form.

$\Rightarrow$ need to approximate

Alternatives to CDF inversion:

- Box-Muller transform: generate U_1, U_2, then

$$\left(\sqrt{-2\log U_1}\sin(2\pi U_2),\ \ \sqrt{-2\log U_1}\cos(2\pi U_2)\right)$$

  are independent standard normal random variables.
- Marsaglia's polar method (similar to Box-Muller)
- Ziggurat algorithm: a type of *rejection sampler*

A Different Simulation Strategy

Thus far, we were directly (by CDF inversion with uniform RNGs) or indirectly (e.g. with importance sampling) generating **independent and identically distributed** variables from the *density of interest*, say f.

In many Bayesian inference problems, there are two frequently-encountered problems which make the above methods difficult or inefficient.

1. Sometimes the problem is very high-dimensional; we can produce multivariate random variates, but it would be useful to have some means of decomposing a high-dimensional problem into a sequence of smaller problems.
2. Often we only know f up to some constant of proportionality, which is non-trivial to compute, and perhaps numerical approximations are costly or inaccurate.

A revolutionary idea: introduce Markov chains, which have some very helpful convergence properties

MCMC idea

We want to sample from some target probability density f, usually a **posterior**, which we can typically write down as $f \propto \text{likelihood} \times \text{prior}$. This is our target.

- Goal: construct a Markov chain in conjunction with likelihood $\times$ prior that has the target f as its stationary distribution, and consider Monte Carlo algorithms which sample from f by utilizing this Markov chain
- a Markov chain is a **random or stochastic process**: a collection of random variables *indexed (and ordered) by time*, where the set of possible states of the process is called the **state space** and each variable takes values on the state space

Markov Chains

A Markov chain $\{X^{(t)}\}$ is a sequence of dependent random variables

$$X^{(0)}, X^{(1)}, X^{(2)}, \ldots, X^{(t)}, \ldots$$

such that the probability distribution of $X^{(t)}$ *given the past variables* depends only on $X^{(t-1)}$.

- this conditional probability distribution is the **transition or Markov kernel** K:

$$X^{(t+1)} | X^{(0)}, X^{(1)}, X^{(2)}, \ldots, X^{(t)} \sim K(X^{(t)}, X^{(t+1)}).$$

Example (Simple Random Walk)

$$X^{(t+1)} = X^{(t)} + \epsilon_t,$$

with $\epsilon_t \sim \mathcal{N}(0,1)$ independently of $X^{(t)}$.
$\Rightarrow K(X^{(t)}, X^{(t+1)}) \equiv \mathcal{N}(X^{(t)}, 1)$

Stationarity

We often will work with Markov chains which exhibit a very strong property called **stationarity**. This means that there exists a probability distribution f such that if $X^{(t)} \sim f$, then $X^{(t+1)} \sim f$.

- this means the transition kernel and stationary distribution must satisfy

$$\int_{\mathcal{X}} K(x, y) f(x) dx = f(y).$$

- the existence of a stationary distribution imposes a constraint on the kernel called irreducibility

Definition

A Markov chain is said to be **irreducible** if, regardless of the starting value $X^{(0)}$, there is a positive probability to eventually reach any part of the state space.

Recurrence and Ergodicity

Definition

A Markov chain is said to be **recurrent** if the chain returns to any arbitrary part of the state space infinitely many times.

For recurrent chains, the stationary distribution is also a limiting distribution.

- this means the limiting distribution of $X^{(t)}$ is f for almost all initial values $X^{(0)}$
- this property is termed ergodicity

Magical result: if a kernel K produces an *ergodic* Markov chain with stationary distribution f, then generating a chain from this kernel K will eventually produce simulations from f

LLN for Markov Chains

Recall that we justified Monte Carlo methods by saying that a suitable LLN assures that the Monte Carlo estimate (the average of the simulated values) converges to the true mean.

For integrable functions h, the average

$$\frac{1}{T}\sum_{t=1}^{T} h(X^{(t)}) \to E_f[h(X)]$$

In this setting, such a result is often called the **Ergodic Theorem**.

Failure of Convergence

We discuss convergence later (time permitting), but it is important to know when convergence never occurs in Bayesian problems: when the posterior is not proper.

Recall: when we use *improper* priors, there is no assurance that the posterior is proper.

- it is often the case that we have no idea of the posterior is proper or not, but we can still utilize MCMC methods to sample from that posterior
- if we're lucky, the Markov chains will diverge quickly and we can see that there is a problem
- however, it often happens that the chain appears stable even for tens or hundreds of thousands of iterations, and we don't let it run long enough to detect problems