Sampling random variables

Stat 580

Motivating example

Suppose you have 1000 dollars. One day, you are given a chance to become rich! You can play the following game repeatedly:

- Decide whether and how much you want to bet on the outcome of a coin toss. Say, your bet is x dollars.
- Toss a *fair* coin
- Then
  - If you bet head and a head shows up, you get $100x$ dollars.
  - If you bet tail and a tail shows up, you get $2x$ dollars.
  - Your bet will be taken away if your guess is incorrect.

Motivating example

As a statistician, you may compute the expected return for a *single* game first.

$$E(\text{" } head \text{ "}) = 49x$$

$$E(\text{" } tail \text{ "}) = 0$$

Wow! What *a* simple game! Bet on head for sure.

- To maximize the expected return, bet all money: $x = 1000$

But it is not *a* game! (Of course, gambling's not a game!)

Well, I meant it is not a single game.

- You can play this game repeatedly.
- Optimize the (expected) long-term return rather!
- If you bet all your money, there is a 0.5 chance that you lose all your money in one single game.

Motivating example

- If you know something related to investment theory or professional gambling, you may have heard of "Fortune's formula" or Kelly's criterion.
    - a formula of x to maximize the long term expected return under the knowledge of the winning probability.
- La vie est dure!
    - Due to certain reason, you are not allowed to spend more than 2000 and less than 500 for each game, whenever you play the game.
    - Should you just use whatever Kelly's criterion determines and trim it up or down to respect the constraint?
- (Stochastic) simulations can help solving complicated problems like this. In exchange, we have to spend computational resources.

Simulation

Suppose we want to know the expected return after 1000 games if a particular strategy is used.

1. Simulate 1000 Bernoulli random variables. Say 1 represents a head and 0 represents a tail.

2. Compute the money we have after this 1000 games, based on that particular strategy.

3. Repeat step 1 and 2 for, say, 100,000 times.

4. Average the 100,000 outcomes.

- Need to know how to simulate a Bernoulli random variables

- for other simulations, we will need to simulate other random variables, not bounded to the standard random variables

Background

Now, assume we can generate from $\text{Unif}(0, 1)$. (See this.)

In R:

```
runif(n)
```

In C:

```
#include <stdio.h>
#include <time.h>
#define MATHLIB_STANDALONE
#include <Rmath.h>

int main() {
  int i;

  set_seed(time(NULL), 580580); /* set seed */

  for (i=1; i<=10; i++){
    printf("%f ", unif_rand());
  }
  return 0;
}
```

Background

```
#include <stdio.h>
#include <time.h>
#define MATHLIB_STANDALONE
#include <Rmath.h>

int main() {
  int i;

  set_seed(time(NULL), 580580); /* set seed */

  for (i=1; i<=10; i++){
    printf("%f ", unif_rand());
  }
  return 0;
}
```

- `unif_rand()` generates a $\mathrm{Unif}(0, 1)$ random variable
- compile with flags "`-lRmath`" and "`-lm`" to link the Rmath and standard math libraries
- try running this program consecutively (within 1 second)
    - `time()` gets the current calendar time as a value of type represented in the number of seconds

Background

An alternative:

```c
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

int main() {
  int i;

  srand(time(NULL)); /* set seed */

  for (i=1; i<=10; i++){
    printf("%f ", rand() /  (double) RAND_MAX );
  }
  return 0;
}
```

Inverse transform method

- applies to univariate random variables
- Let F be a distribution function and define $F^{-1}(u) = \inf\{x : F(x) \geq u\}$.

  Theorem: If $U \sim \text{Unif}(0, 1)$, then $F^{-1}(U) \sim F$.

Example (exponential random variable)

The CDF of $\text{Exp}(\theta)$ is

$$F(x) = 1 - \exp\left(-\frac{x}{\theta}\right), \quad x \geq 0, \theta > 0.$$

- Since $F^{-1}(u) = -\theta \log(1 - u)$ for $u \in (0, 1)$, then $-\theta \log(1 - U) \sim \text{Exp}(\theta)$ where $U \sim \text{Unif}(0, 1)$.
- Since $1 - U \sim \text{Unif}(0, 1)$, then also $-\theta \log U \sim \text{Exp}(\theta)$.

Example (exponential random variables)

```
#include <stdio.h>
#include <time.h>
#define MATHLIB_STANDALONE
#include <Rmath.h>
#define theta 1

int main() {
  int i;
  double u, x;

  set_seed(time(NULL), 580580); /* set seed */

  for (i=1; i<=10; i++){
    u = unif_rand(); /* uniform random variable */
    x = -theta * log(u);
    printf("%f ", x);
  }
  return 0;
}
```

Example (discrete random variables)

- Let X be a discrete random variable with values in $\{c_1, c_2, \dots, c_n\}$.
- Let

$$q_0 = 0 \quad q_i = \sum_{j=1}^{i} p(X = c_j) \quad i = 1, \dots, n.$$

- To sample X:
  1. Generate $U \sim \text{Unif}(0, 1)$.
  2. Find $k \in \{1, \dots, n\}$ such that $q_{k-1} < U < q_k$.
  3. Set $X = c_k$.
- If $c_1 < \cdots < c_n$, it can be derived from the inverse transform method.
- But this algorithm also works even if these c_i's are not sorted.
- This algorithm can be extended similarly to countably infinite number of c_i's.

Sampling from a truncated distribution

- Let $X \sim F$. We want to sample X conditional on $a < X \leq b$.
- Recall $P(x \in (a, b]) = F(b) - F(a)$ and assume $F(b) > F(a)$.

  Theorem: Let $A = F(a), B = F(b)$. Then $F^{-1}\{A + (B - A)U\}$ follows the conditional distribution.

  Proof :

$$
\begin{aligned}
P\{F^{-1}(A + (B - A)U) \leq x\} &= P\{A + (B - A)U \leq F(x)\} \\
&= P\left\{ U \leq \frac{F(x) - F(a)}{F(b) - F(a)} \right\} \\
&= \frac{F(x) - F(a)}{F(b) - F(a)} \\
&= P(X \leq x | a < X \leq b).
\end{aligned}
$$

Example (Truncated exponential distribution)

- Let $c > 0$ and $X \sim \text{Exp}(\theta)$.
- Goal: sample X conditional on $X \geq c$
- First method: Use $F^{-1}\{F(c) + (1 - F(c))U\}$
- Second method:
  - Recall the exponential distribution is memoryless in the following sense:

$$X - c \sim \text{Exp}(\theta_0) \text{ given } X > c$$

  - Therefore, the conditional distribution can be sampled by $-\theta \log U + c$ with $U \sim \text{Unif}(0, 1)$.

Numerical evaluation of inverse transform

- What do we do when F^{-1} is not known explicitly?
- For continuous random variable, computing $F^{-1}(u)$ is equivalent to finding a root of x of the equation $F(x) - u = 0$.
- Let f be the density of X. Newton's method gives

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

- Under suitable conditions, $x_n \to F^{-1}(u)$.

Rejection sampling

- Want to sample from a pmf/pdf $f(x)$ defined on $\mathcal{X}$.
- Suppose we know $f(x)$ is proportional to a function $q(x)$; i.e., $f(x) = cq(x), x \in \mathcal{X}$.
    - $c = \left\{\int_{\mathcal{X}} q(x)dx\right\}^{-1}$ maybe unknown or hard to evaluate.
    - fine as long as we know $f(x)$ up to a constant, common in Bayesian analysis (posterior distributions)
- Let $g(x)$ be a density defined on $\mathcal{X}$, and we know how to generate from $g(x)$.
- Further suppose that, for some $\alpha > 0$, $q(x) \leq \alpha g(x)\ \forall x \in \mathcal{X}$.
- The function $\alpha g(x)$ is known as the *envelope*.

Rejection sampling

- The algorithm is as follows:
  1. Sample $X \sim g(x), U \sim \text{Unif}(0,1)$ independently.
  2. If $U > \frac{q(X)}{\alpha g(X)}$, then go to step 1, otherwise return X. The returned value is a random variable from $f(x)$.
- Sketch of the proof: Denote $r(x) = \frac{q(x)}{\alpha g(x)}$ and note that $r(x) \in [0,1]$. Let Y be a sample returned by the algorithm. For any $A \subset \mathcal{X}$,

$$P(Y \in A) \overset{\text{why?}}{=} P\{X \in A | U \leq r(X)\} = \frac{P\{X \in A, U \leq r(X)\}}{P\{U \leq r(X)\}}$$

  We verify that $P(Y \in A) = \int_A f(x)dx$ by showing (in homework) that
  - acceptance probability p_a: $P\{U \leq r(X)\} = \frac{1}{\alpha}\int_{\mathcal{X}} q(x)dx$
  - $P\{X \in A, U \leq r(X)\} = \frac{1}{\alpha}\int_A q(x)dx$

Rejection sampling

- The probability of acceptance in each iteration is $p_a = \frac{\int_{\mathcal{X}} q(x)dx}{\alpha}$.
  - For efficiency, we want this probability to be large.
- The number of iterations until an acceptance is geometrically distributed with mean $\frac{1}{p_a}$.
- Given g, the optimal α to maximize p_a is $\alpha = \sup \frac{q(x)}{g(x)}$.
- We want to have p_a close to 1, which requires a good choice of $g(x)$.

Example (Beta distribution)

For $a, b > 0$, $\text{Beta}(a, b)$ has density

$$f(x) = \frac{x^{a-1}(1-x)^{b-1}}{B(a,b)} \mathbf{1}_{\{0 \le x \le 1\}}.$$

- Rejection sampling for the case $a, b > 1$ (i.e. the density is bounded):
  - Choose $g(x)$ as $\text{Unif}(0, 1)$ and set $f(x) = q(x)$.
  - Note $f(x)$ is maximized at $x_0 = \frac{a-1}{a+b-2}$.
  - Therefore, our optimal α is $\alpha = \sup \frac{f(x)}{g(x)} = f(x_0)$.
  - The algorithm becomes:
    1. Draw $X, U \sim \text{Unif}(0, 1)$ until $f(x_0)U \le f(X)$
    2. Return X.

Example (Beta distribution)

- A faster and more general alternative ($a > 0, b > 0$):
  - Fact: for independent $Y \sim \text{Gamma}(a, 1)$ and $Z \sim \text{Gamma}(b, 1)$, $\frac{Y}{Y+Z} \sim \text{Beta}(a, b)$.
  - $\text{Gamma}(a, 1)$ is defined as $\text{Exp}(a)$.
    - we know how to generate exponential random variable.

Rejection sampling - conditional distributions

- Let A be a subset of $\mathcal{X}$. To sample X conditional on $X \in A$, one can use the following crude rejection procedure:

  1. Sample X until $X \in A$.

  2. Return X.

- Sometimes more carefully designed rejection sampling can lead to a faster algorithm.

  - For example:

    - Let $c > 0$ and c is large.

    - To sample X from $\mathcal{N}(0, 1)$ conditional on $X \geq c$, the simple rejection sampling is very inefficient. Why?

Example

- To improve efficiency, an exponential distribution can be used as the envelope.
- It suffices to sample $X - c$:
  - For $X \sim \mathcal{N}(0, 1)$, the conditional density of $X - c | X \geq c$ is
    $$f(s) = \frac{\phi(s + c)}{1 - \Phi(c)}, s \geq 0.$$
  - For $Y \sim \text{Exp}(\lambda)$, the conditional density of $Y - c | Y \geq c$ is
    $$g(s) = \lambda \exp(-\lambda s), s \geq 0.$$

    (memoryless property)

Example

- Given λ, the optimal α is

$$\begin{aligned}\alpha &= \sup_{s\geq 0} \frac{\phi(s+c)/\{1-\Phi(c)\}}{\lambda\exp(-\lambda s)} \\ &= \sup_{s\geq 0} \left[\frac{\exp(\frac{1}{2}\lambda^2 - \lambda c)}{\sqrt{2\pi}\lambda\{1-\Phi(c)\}} \right]\end{aligned}$$

- The value that maximizes the expression inside the square brackets is

$$\lambda = \frac{1}{2}(c + \sqrt{c^2+4}).$$

- Thus, the optimal p_a is

$$\frac{1}{\alpha} = \frac{\sqrt{\pi}(c+\sqrt{c^2+4})\{1-\Phi(c)\}}{\sqrt{2e}} \geq 0.$$

Univariate Normal Distribution

- Let Φ be the CDF of $\mathcal{N}(0, 1)$.
- To sample $X \sim \mathcal{N}(0, 1)$, one can use the Box-Müller method.
    1. Sample U_1, U_2 iid $\sim$ $\text{Unif}(0, 1)$.
    2. Set $R = \sqrt{-2 \ln U_1}$.
    3. Return $Z_1 = R\cos(2\pi U_2)$ and $Z_2 = R\sin(2\pi U_2)$.
- Here Z_1 and Z_2 are iid $\sim \mathcal{N}(0, 1)$.

Rmath library

- Generation of standard random variables

```
double rnorm(double mu, double sigma);
```

- Distribution functions of standard random variables

```
double dnorm(double x, double mu, double sigma, int give_log);
double pnorm(double x, double mu, double sigma,
             int lower_tail, int give_log);
double qnorm(double p, double mu, double sigma,
             int lower_tail, int log_p);
```

- Various mathematical functions

```
double gammafn(double x);
double choose(double n, double k);
```

- Various mathematical constants

```
M_E   /* e */
M_PI  /* pi */
```

Example

```
#include<stdio.h>
#include<time.h>
#define MATHLIB_STANDALONE
#include<Rmath.h>

int main(){
  double mu, sigma, prob;
  time_t t;

  printf("Enter the mean: ");
  scanf("%lf", &mu); /* new input function */

  printf("Enter the sd: ");
  scanf("%lf", &sigma);

  printf("Enter the prob. level: ");
  scanf("%lf", &prob);

  printf("Answer: %f\n", qnorm(prob, mu, sigma, 1, 0));

  t = time(NULL);
  set_seed(t, 77911);

  printf("generated normal random variable: %f\n", rnorm(mu, sigma));

  return 0;
}
```

Multivariate Normal Distribution

- Let $\boldsymbol{\mu} \in \mathbb{R}^d$ and $\boldsymbol{\Sigma} \in \mathbb{R}^{d \times d}$ be nonnegative definite.
- Recall for $\boldsymbol{X} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and $\boldsymbol{A} \in \mathbb{R}^{n \times d}$,

$$\boldsymbol{A}\boldsymbol{X} \sim \mathcal{N}(\boldsymbol{A}\boldsymbol{\mu}, \boldsymbol{A}\boldsymbol{\Sigma}\boldsymbol{A}^T).$$

- To sample $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, first compute the Cholesky decomposition of $\boldsymbol{\Sigma}$: $\boldsymbol{\Sigma} = \boldsymbol{A}\boldsymbol{A}^T$ where $\boldsymbol{A}$ is lower triangular ($\boldsymbol{A}$ is sometimes called the square root of $\boldsymbol{\Sigma}$)
- Set $\boldsymbol{X} = \mu + \boldsymbol{A}\boldsymbol{Z}$ where coordinates of $\boldsymbol{Z}$ are iid $\mathcal{N}(0, 1)$.
- We need arrays and some linear algebra! How to generate normal random vector in C?