Generating Random Numbers

JMR Ch 18

BTRY/STSCI 4520

(Note Chs 13 - 17 Assumed Knowledge, but a good reference)

Where Do Random Numbers Come From?

- Generated from physical processes (background radiation, radio-active decay etc)
  - Fairly impractical in a computer

- External generation (human mouse movements, etc...)
  - Hard to model.
- Computer generated *pseudo-random* numbers

  - Deterministic (you'll get the same answer from the same starting point), but looks close to random.

Congruential Generators

Simplest effective pseudo-random number generator

$$X_{n+1} = (AX_n + B)(\mod m)$$

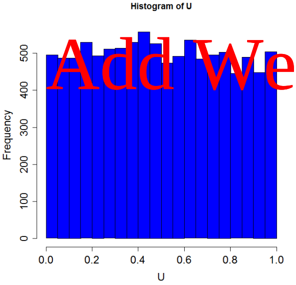
- results in numbers between 0 and m.
- Will only repeat after m steps if m and B are relatively prime and $A-1$ is divisible by prime factors of m.

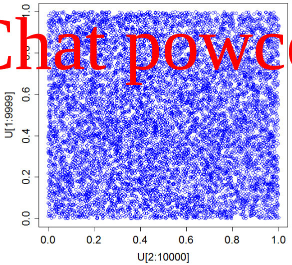
- To obtain uniform pseudo-random numbers take X_n/m.
- Need to be cautious; can detect a deterministic relationship.
- **But** determinism can also be helpful (see later).

Example

JMR recommends

```
X[1] = 3

A = 1664525
B = 1013904223
m = 2^(32)

for(i in 2:10000){ X[i] = (A*X[i-1]+B)%%m }
```



But Take Care

The RANDU generator was shipped with Unix systems in the 1970s, using


```
A = 65539
B = 0
m = 2^(31)
```

Plotting 3-dimensional runs (X_{t-1}, X_t, X_{t+1}) together reveals evident patterns – see R code for lecture.

- Power of 2 used for m because convenient for binary arithmetic (very low-level computing).
- RANDU chosen also for convenience – problems detected because simulation results did not match theory.
- Period is $2^{32} = 4,294,967,296$ before repeating numbers; usually enough.

In R

- Congruential generators a good first start, now somewhat obsolete.
- Observable correlation between X_k and X_{k+r} (eg as in RANDU) and relatively short periods.
- But require little memory, very fast to use.
- New methods based on manipulating *bits* of binary representation for X_k.
- R uses the *Mersenne-Twister* (developed 1997) which acts along these lines
  - Period is $2^{19937} - 1$ (not storable in R).
  - Plots of 623-dimensional runs (if you can think of this) still look uniform.

Seeds and Repeatability

- Pseudo-random number generators are *deterministic*: if you start them in the same place, you get the same answer.
- Typically R chooses a seed as a starting point; often from system clock.
- But, you can specify this with an integer giving where in the generator's cycle you want to start:

```
> runif(5)
[1] 0.6223777 0.6754986 0.8022900 0.2603083 0.7597607
> set.seed(36)
> runif(5)
[1] 0.6223777 0.6754986 0.8022900 0.2603083 0.7597607
> runif(5)
[1] 0.01990291 0.95542781 0.43666244 0.08922046 0.36051
```

- Instead of storing everything in a simulation, this lets you re-run it *exactly*.
- Often simulation time mitigates against this, but it can be convenient.

R and Seeds

- Besides `set.seed`, R also stores `.Random.seed`.
- This is a vector of 626 integers that also specify parts of the random number generator.
- Usually remains constant (across R sessions and computers), but saving it can ensure compatibility over platforms.


```
> RNG.seed = .Random.seed
> runif(5)
[1] 0.80228995 0.26030829 0.75976074 0.01990291 0.95542781
> .Random.seed = RNG.seed
> runif(5)
[1] 0.80228995 0.26030829 0.75976074 0.01990291 0.95542781
```

Also doesn't require you to make up an integer. Works for any simulation (as long as you do exactly the same commands).

From Uniform to Discrete Random Variables

From here on assume we can generate $U(0,1)$ random variables – how do we get to others?

- For Bernoulli random variables with $B(p)$, then if $U \sim U(0,1)$,

$$P(U < p) = p$$

so take $X = I(U < p)$.

- More generally, if X has integer values, consider the cumulative distribution function

$$F(n) = \sum_{i=1}^{n} P(X = i).$$

We can generate X by taking $U \sim U(0,1)$ and

$$X : F(X-1) < U \leq F(X)$$

Then

$$P(F(X-1) < U \leq F(X)) = F(X) - F(X-1) = P(X)$$

Example

Simulating from a Poisson:

```
U = runif(1)
X = 0
while( ppois(X,3) < U){ X = X+1 }
```

See code simulation to check that this produces the right distribution.

Often $F(X)$ is not easy to calculate, but $p(X)$ is; note we can update $F(X)$ within the while loop:

```
U = runif(1)
X = 0
FX = dpois(0,3)
while( FX < U){ X = X+1; FX = FX + dpois(X,3) }
```

`dpois` much cheaper than `ppois` to calculate.

Some Special Cases

There are often constructive definitions of r.v.'s that can be employed.

- Binomial random variables are a sum of independent Bernoulli's: $X \sim Bin(n,p) \Rightarrow X = \sum_{i=1}^{n} Z_i$ where $Z_i \sim B(1,p)$

```
n = 10
p = 0.45
X = sum( runif(n) < p )
```


- Geometric or negative binomials – see exercises from Lecture 2.
- Uniform on the integers $1,\dots,N$; use the `ceil` command


```
> N = 100; n = 10;
> ceil( N*runif(n) )
 [1] 75 51 13 27 92 20 45 20  8 61
```

- We can now generate bootstrap samples:

```
I = ceil( nrow(faithful)*runif(nrow(faithful)) )
faithboot = faithful[I,]
```

Generating Permutations

Not so simple, because we only want each element once.

1. Select one item in turn and add it to the new set.
2. Remove that item from the set to be selected.


```
N = 10; I = 1:N; Iperm = rep(0,N)
for(i in 1:N){
  k = ceil( length(I)*runif(1) )
  Iperm[i] = I[k]
  I = I[-k]
}
```


You could also do this by swapping elements.

Continuous Random Variables

The *inversion* method is the simplest way to generate continuous random variables

We know that if X has cumulative distribution function F, then $F(X) \sim U(0,1)$, or in other words that X has the same distribution as $F^{-1}(U)$

$$P(F^{-1}(U) \le x) = P(U \le F(x)) = F(x)$$

Only problem is that $F^{-1}(x)$ easy to obtain only in special cases.

Important Special Cases

Uniform $U(a, b)$ Density: $I(x \in [a\ b])/(b-a)$

cdf $F(x) = (x-a)/(b-a)$ if $a \leq x \leq b$

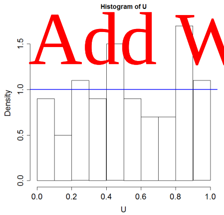
inverse cdf $F^{-1}(U) = (b-a)U + a$

Exponential $exp(\lambda)$ Density $\lambda e^{-\lambda x}$

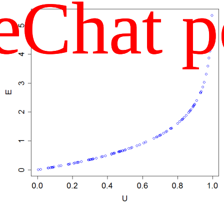
cdf $F(x) = 1 - e^{-\lambda x}$ if $x > 0$

inverse cdf $F^{-1}(U) = -\lambda^{-1}\log(1-U)$

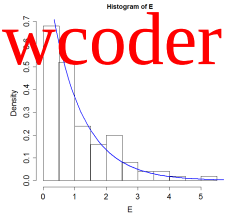
Uniform



Inverse CDF



Exponential(1)

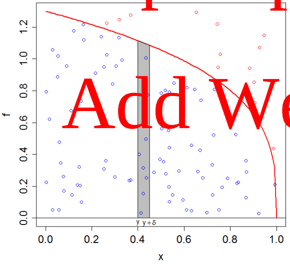


Rejection Method

When F^{-1} is not easy to calculate explicitly – could try numerically.

Alternatively, rejection method is sometimes faster. Simplest case:

- $X \sim f(\cdot)$ with support on $[a, b]$ and $f(x) \leq k$.
- Generate $Y \sim U(a, b)$ and $Z \sim U(0, k)$.
- Set $X = Y$ if $Z < f(Y)$, otherwise try again.



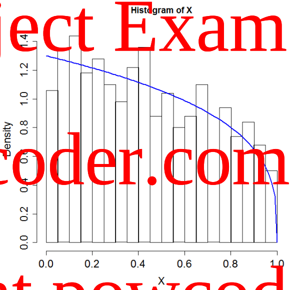
$$P(y < X < y + \delta) = \frac{\int_y^{y+\delta} f(x)dx}{\int_a^b f(x)dx} = \int_y^{y+\delta} f(x)dx$$

Because Y, Z uniform on the square.

In Code

We'll use a *Beta*$(1, 1.3)$ distribution. This has maximum value 1.3.


```
X = rep(0,1000)
for(i in 1:1000){
  Accept = FALSE
  while(!Accept){
    Y = runif(1)
    Z = runif(1,0,1.3)
    if( Z < dbeta(Y,1,1.3) )
      Accept = TRUE
  }
  X[i] = Y
}
```

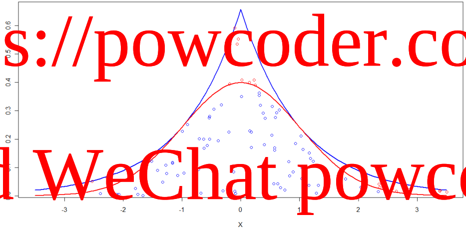


Cheaply (no fixed n):

```
Y = runif(1000)
Z = runif(1000,0,1.3)
Accept = Z < dbeta(Y,1,1.3)
X = Y[Accept]
```

Generalized Rejection Method

- For densities on the whole real line, we can't use a uniform distribution.
- But if we can simulate from $h(x)$ where $kh(x) \geq f(x)$ we can use the same ideas.



Call $kh(x)$ the *envelope* for $f(x)$.

1. Generate $Y \sim h(\cdot)$
2. Generate $Z \sim U(0, kh(Y))$
3. Accept Y if $Z < f(Y)$

Justification

General rejection method is justified because the (Y, Z) pairs are uniformly distributed over the region below $kh(x)$.

$$
\begin{aligned}
&P((Y,Z) \in (y, y+dy) \times (z, z+dz)) \\
&\quad = P(Z \in (z, z+dz) | Y \in (y, y+dy)) P(Y \in (y, y+dy)) \\
&\quad = \frac{dz}{kh(y)} h(y) dy \\
&\quad = dzdy/k
\end{aligned}
$$

because $P(Z \in (z, z+dz) | Y \in (y, y+dy))$ is uniform on $[0 \; kh(y)]$.

This means the points we accept are uniformly distributed on the region under $f(x)$ and therefore the x-coordinates have density $f(x)$.

Example

- In picture above, we used a Laplace distribution $h(x) = \frac{1}{2}e^{-|x|}$ as an envelope for standard normal.

- To generate from Laplace, use $V \sim B(0.5)$ and $U \sim U(0,1)$, then $2(V - 0.5) \in \{-1, 1\}$ and $\log U \sim exp(1)$ so

$$2(V - 0.5)\log U \sim h(\cdot).$$

- To find k, ratio of densities is

$$\frac{\frac{1}{\sqrt{2\pi}}e^{-x^2/2}}{\frac{1}{2}e^{-|x|}} = \frac{\sqrt{2}}{\sqrt{\pi}}e^{|x|-x^2/2} \leq \sqrt{\frac{2e}{\pi}}$$

Note: JMR does 1/2 normal, and then uses $2(V - 0.5)$ to symmetrize.

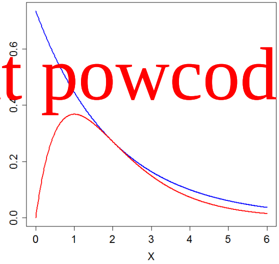
Example Continued

We'll fix the size of Y and Z and just see how many X we get after rejection:


```
V = 2*((runif(1000)>0.5)-0.5)   # Generate Laplace r.v.'s
U = runif(1000)
Y = V*log(U)

# Uniforms
Z = runif(100)*exp(-abs(Y))*sqrt(2*exp(1)/pi)

# Which are accepted?
Accept = Z < dnorm(Y)

# Now we get our sample
X = Y[Accept]
```

Efficiency

- In last example above, we accept about 75% of tries.
- Higher acceptance probability = less computational work.
- Over-all acceptance rate = $1/k$ (area under $kh(x) = k$, area under $f(x) = 1$).
- Alternatively, number tries before accepting is Geometric$(1/k)$ with expected value k.
- Two things we can affect
  - Choice of $h(x)$ (harder)
  - Choice of k.
- See optimizing Gamma in book (and on the board).



Normal Random Variable Methods

Note if $X \sim N(0,1)$, then $\sigma X + \mu \sim N(\mu, \sigma^2)$, easy once we can generate $N(0,1)$.

1. Improved rejection methods from above
   - In fact, $Z \sim U(0, \sqrt{2e/\pi} e^{\log U}) = U(0, U\sqrt{2e/\pi})$
   - We can also throw away V and just decide to make Y negative 1/2 time based on Z.

   1. $U \sim U(0,1)$, $Z \sim U(0, U\sqrt{2e/\pi})$, $Y = \log(U)$
   2. If $Z < \phi(Y)/2$ return $-Y$, if $\phi(Y)/2 < Z < \phi(Y)$ return Y, otherwise repeat.

2. Central limit theorem: $\mathrm{Var}(U) = 1/12$, so

$$\left(\sum_{i=1}^{12} U_i \right) - 6 \approx N(0,1).$$

12 is a bit small; could add more terms + rescale, but this is computationally expensive.

Constructive Methods

Remember, direct transforms are generally fastest.

Exponential	$-(\log U)/\lambda$ if $U \sim U(0,1)$.
$B(p)$	$I(U < p)$ if $U \sim U(0,1)$.
Laplace	$2(V - 1/2)E$ if $E \sim$ exponential and $V \sim B(0.5)$.
$B(n,p)$	$\sum_{i=1}^{n} Z_i$ if $Z_1, \ldots, Z_n \sim B(p)$.
χ^2_d	$\sum_{i=1}^{d} X_i^2$ if $X_1, \ldots, X_d \sim N(0,1)$.
$F^{d_1}_{d_2}$	$(Z_{d_1}/d_1)/(Z_{d_2}/d_2)$ if $Z_{d_1} \sim \chi^2_{d_1}$ and $Z_{d_2} \sim \chi^2_{d_2}$.

Many many other relationships; some derived, some constructed.

Box-Muller for Gaussians

- Clever construction method. Look at *pairs* of Gaussians (X, Y).
- Now in polar co-ordinates (R, Θ), we have $R \sim exp(0.5)$, $\Theta \sim U(0, 2\pi)$. (first is not obvious)

- So if we have $U_1, U_2 \sim U(0, 1)$, then $\sqrt{-2\log U_1}\cos(2\pi U_2)$ and $\sqrt{-2\log U_1}\sin(2\pi U_2)$ are Gaussian!

This yields the following

1. $U_1, U_2 \sim U(0, 1)$
2. $X = \sqrt{-2\log U_1}\cos(2\pi U_2)$, $Y = \sqrt{-2\log U_1}\sin(2\pi U_2)$

To obtain independent normal $X, Y \sim N(0, 1)$.

More Efficient Box-Muller

Trigonometric functions are expensive.

- Instead, if (A, B) uniform on the circle with polar coordinates (S, Ψ), $S^2 = A^2 + B^2 \sim U(0,1)$ (again not obvious).
- So that $(\sqrt{-2\log S^2}, \Psi)$ has the same distribution as (R, Θ).
- Also $\cos(\Psi) = A/S$, $\sin(\Psi) = B/S$.

Improved algorithm is

1. $U, V \sim U(-1, 1)$ (uniform on box)
2. If $S^2 = U^2 + V^2 > 1$ retry (rejection: uniform on circle)
3. Set $W = \sqrt{(-2\log S^2)/S^2}$
4. $X = UW$, $Y = VW$.

Summary

- Random number generation actually *pseudo*-random.
- But deterministic random variables allow you to repeat simulations easily.

- Once you can get uniform random variables, get others by
  1. transforms
  2. rejection methods
  3. being very clever

- Next: Monte Carlo integration.