Generating random numbers

Generating uniform(0,1) deviates

Books: DE Knuth (1998) The art of computer programming, vol 2, 3rd ed, ch 3
Numerical recipes in C, ch 7

Linear congruential generator

$$X_{n+1} = a X_n + c \pmod m$$

m = modulus = $2^{32} - 1$
a = multiplier = choose carefully!
c = increment = (maybe) 0
X_0 = seed

- $X_n \in \{ 0, 1, 2, \ldots, m - 1 \}$; $U_n = X_n/m$
- Use odd number as seed
- Algebra/group theory helps with choice of a
- Want **cycle** of generator (number of steps before it begins repeating) to be **large**
- Don't generate more than m/1000 numbers

Composite generator

$X_{n+1} = a_1 X_n + c_1 \pmod m$
$Y_{n+1} = a_2 Y_n + c_2 \pmod m$

$W_{n+1} = X_n + Y_n \pmod m$

Shuffling a random number generator

- **Initialization**
  - Generate an array R of n (= 100) random numbers from sequence $\langle X_k \rangle$
  - Generate an additional number X to start the process
- **Each time generator is called**
  - Use X to find an index into the array R
    $j \leftarrow \lfloor X * n \rfloor$
  - $X \leftarrow R[\, j \,]$
  - $R[\, j \,] \leftarrow$ a new random number
  - Return X as random number for call

Shuffling with two generators

- **Initialization**
  - Generate an array R of n (= 100) random numbers from sequence $\langle X_k \rangle$
- **Each time generator is called**
  - Generate X from $\langle X_k \rangle$ and Y from $\langle Y_k \rangle$
  - Use Y to find an index into the array R
    $j \leftarrow \lfloor Y * n \rfloor$
  - $Z \leftarrow R[\, j \,]$
  - $R[\, j \,] \leftarrow X$
  - Return Z as random number for call

Generating random numbers in R

`.Random.seed`

changed or created after each call to any of the random number generators

`runif(n, min=0, max=1)`

generates uniform(0,1) random numbers

Access from C

```
#include <R.h>

GetRNGstate();
PutRNGstate();

double unif_rand();
double norm_rand();
double exp_rand();
double r****();
```

```
/usr/local/lib/R/include/R.h
/usr/local/lib/R/include/R_ext/Mathlib.h
```

```
RHOME/src/main/RNG.c
RHOME/src/nmath/snorm.c
RHOME/src/nmath/sexp.c
```

Random draw from { 1, 2, 3, …, n } with probabilities p_1, p_2, p_3, …, p_n

```
r <- runif(1)
for(i in 1:n) {
   if(r < p[i]) return(i)
   else r <- r – p[i]
}

sample(1:n, 1, prob=p)
```

Random permutation of {1, 2, …, n }

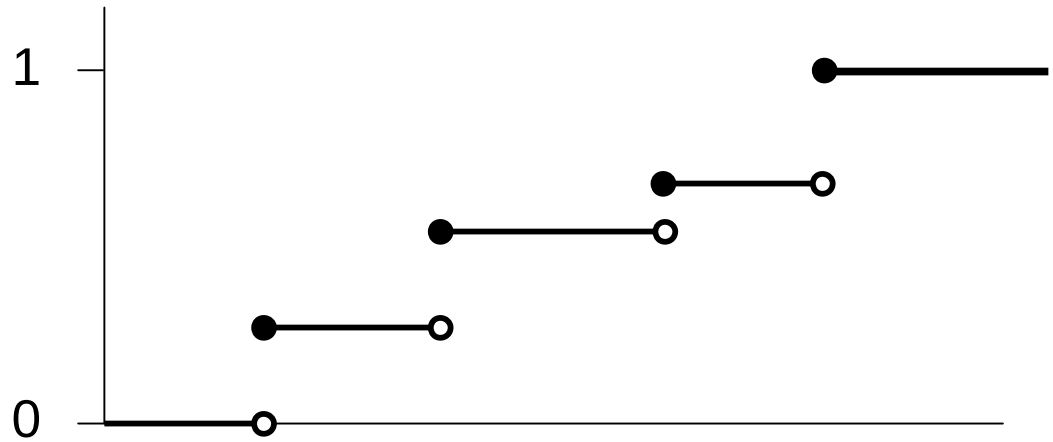
```
for(i in 1:n) {
   r <- sample(i:n, 1)
   # exchange x[i] and x[r]
   x[c(i,r)] <- x[c(r,i)]
}

sample(1:n)
```

Simulate r.v. X with cdf F

U ~ uniform(0,1)

X ~ cdf F, so $\Pr(X \le x) = F(x)$; let $G = F^{-1}$

G(U) has the same distribution as X



Example: exponential(λ)

$F(x) = 1 - \exp(-\lambda x)$

$F^{-1}(u) = -\log(1 - u) / \lambda$

U ~ uniform(0,1)

$X = -\log(U) / \lambda$ ~ exponential (λ)

Geometric(p)

E ~ exponential[$\lambda = -\log(1-p)$]

X = $\lfloor E \rfloor$ or $\lfloor E \rfloor + 1$

Generating Gaussian deviates

CLT method

Generate $U_1, U_2, \ldots, U_n \sim$ uniform(0,1)

$X = (\sum U_i - n/2) / \sqrt{(n/12)} \sim$ normal(0, 1)

Polar method (Marsaglia 1962)

1. Generate $U_1, U_2 \sim$ uniform(0,1)
2. Calculate $V_i = 2U_i - 1$ [$V_i \sim$ uniform(-1, 1)]
3. Calculate $S = (V_1)^2 + (V_2)^2$
4. If $S \geq 1$, return to step 1
5. Let $Z = \sqrt{[-2 \log(S) / S]}$
6. $X_1 = V_1 Z$ $X_2 = V_2 Z$

Multiple calls: use a *static* variable

1. Generate V_1 and V_2; return V_1
2. Return V_2
3. Generate V_1 and V_2; return V_1
4. Return V_2

Acceptance - rejection technique (Von Neumann 1951)

We wish to simulate from the density f(x)

Suppose f(x) is *majorized* by g(x):

$\exists$ c > 1 such that $f(x) \leq c\, g(x) = h(x)$ for all x

1. Sample X from g(x)
2. Sample U ~ uniform(0,1)
3. Accept X if $U \leq f(X) / h(X)$
   [otherwise, return to 1.]

It's often convenient to use exponentials for this.

Multivariate normal

Suppose we wish to generate

$X = (x_1, \ldots, x_p)' \sim MVN(\mu, \Sigma)$

Use $Z = (z_1, \ldots, z_p)' \sim$ iid normal(0,1)

var(D'Z) = D' var(Z) D = D'D

1. Cholesky decomposition $\Sigma = D' D$
2. $Z = (z_1, \ldots, z_p)' \sim$ iid normal(0,1)
3. $X = D'Z + \mu$

```
rmvn <-
function(n, mu=0, V = matrix(1))
{
  p <- length(mu)
  if(any(is.na(match(dim(V),p))))
    stop("Dimension problem!")

  D <- chol(V)

  matrix(rnorm(n*p), ncol=p) %*% D +
    rep(mu,rep(n,p))
}
```

Order statistics (K Lange's example 20.7.5)

Exponential distribution

Consider $X_1, X_2, \ldots, X_n \sim$ iid exponential($\lambda = 1$)

Order statistics $X_{(1)}, X_{(2)}, \ldots, X_{(n)}$

Define $Z_1 = X_{(1)}$; $Z_{i+1} = X_{(i+1)} - X_{(i)}$

Then the Z_i are independent; $Z_i \sim \exp(n - i + 1)$

$$X_{(k)} = Z_1 + Z_2 + \ldots + Z_k$$

Uniform distribution

$$U_{(k)} = \exp(- X_{(n - k + 1)})$$

or $$U_{(k)} = 1 - \exp(- X_{(k)})$$

Empirical distribution $\{ Y_1, Y_2, \ldots, Y_n \}$

Let $j = \lfloor n U_{(k)} \rfloor$

$Y^*_{(k)} = Y_{(j)}$

NOTE: This last bit is a Sunday night calculation, and so should be treated with caution!

Order statistics from empirical distribution

```
ordstat1 <-
function(x, k=floor(length(x)*0.95))
sort(sample(x,repl=T))[k]

ordstat2 <-
function(x, k=floor(length(x)*0.95))
{
   # x <- sort(x)
   n <- length(x)
   x[ceiling(n*exp(-sum(
      rexp(n-k+1, n-(1:(n-k+1))+1)
         ))))]
}

> x <- rgamma(1000, 5, 5)
> u1 <- u2 <- 1:10000

> unix.time( for(i in 1:10000)
+   u1[i] <- ordstat1(x,950) )
[1] 170.34 2.59 172.98 0.00 0.00

> unix.time( for(i in 1:10000)
+   u2[i] <- ordstat2(x,950) )
[1] 8.32 0.50 8.82 0.00 0.00
```