Solutions to Selected Exercises for Braun and Murdoch's A First Course in Statistical Programming with R

Kristy Alexander, Yiwen Diao, Qiang Fu, and Yu Han
W. John Braun and Duncan J. Murdoch

November 1, 2007

Chapter 5

Simulation

5.2 Generation of Pseudorandom Numbers

1.
```
> x0 <- 17218
> x <- numeric(20)
> x[1] <- (172 * x0) %% 30307
> for(i in 2:20){
+  x[i] <- (x[i-1] * 172) %% 30307
+  }
> x
 [1] 21717  7563 27942 17518 12703  2812 29059 27800 23401 24448 22690 23384
[13] 21524  4674 15946 15082 18009  6234 11503  8561
```

3. (a)
```
> set.seed(32078)
> runif(20,0,1)
 [1] 0.25646258 0.49881767 0.52665491 0.62698163 0.80527541 0.18434520
 [7] 0.51023267 0.36839051 0.17081759 0.74328879 0.01424726 0.22347861
[13] 0.36032442 0.79655794 0.01565888 0.40551753 0.65151347 0.91013292
[19] 0.77073083 0.51094885
```

(b)
```
> set.seed(32078)
> runif(20,3,7)
 [1] 4.025850 4.995271 5.106620 5.507927 6.221102 3.737381 5.040931 4.473562
 [9] 3.683270 5.973155 3.056989 3.893914 4.441298 6.186232 3.062636 4.622070
[17] 5.606054 6.640532 6.082923 5.043795
```

(c)
```
> set.seed(32078)
> runif(20,-2,2)
 [1] -0.974149697 -0.004729333  0.106619657  0.507926506  1.221101642
 [6] -1.262619189  0.040930690 -0.526437979 -1.316729628  0.973155177
[11] -1.943010969 -1.106085563 -0.558702309  1.186231747 -1.937364492
[16] -0.377929887  0.606053870  1.640531669  1.082923308  0.043795402
```

5. (a)
```
> r <- runif(10000, 3.7, 5.8)
> mean(r)
[1] 4.756022
> var(r)
[1] 0.3664445
> sd(r)
[1] 0.6053466
```

(b) `> length(r[r>4])/length(r)`

```
[1] 0.8584
```

7.
```
> U1 <- runif(10000)
> U2 <- runif(10000)
> U3 <- runif(10000)
> U <- U1+U2+U3
```

(a) `> mean(U)`

```
[1] 1.50653
```

(b) `> var(U)`

```
[1] 0.2548007
```

```
> var(U1)+var(U2)+var(U3)
```

```
[1] 0.251535
```

(c) `> mean(sqrt(U))`

```
[1] 1.207929
```

(d)
```
> V <- sqrt(U1)+sqrt(U2)+sqrt(U3)
> length(V[V>=.8])/length(V)
```

```
[1] 0.9965
```

9. (a) `> sample(1:100, size=50, replace=F)`

```
 [1] 26 71 83 86 60 37 85 97 34 33 25 91 78 15 74 69 84 82 63 64 35 40 73 38 16
[26] 32 22 43 19 88 68 12  8 80 17 81 99  4 66 79 49 11 31  1 61  7 46 27 39  9
```

(b) `> sample(1:100, size=50, replace=T)`

```
 [1] 80 32 67 86 77 87 48 41 85 53 29 17 76 72 66 56 32 36 17 61 27 87  8  2 18
[26] 42 88 99 60 80 38 25 68 79 98  9 40 34 16 72 18 87 15 65  8 65  3 90 15 64
```

5.3 Simulation of Other Random Variables

5.3.1 Bernoulli random variables

1. (a)
```
> r <- rbinom(n=10, size=1, p=.5)
> sum(r)
[1] 6
```
   (b)
```
> r <- rbinom(n=1000, size=1, p=.5)
> sum(r)
[1] 481
```

3.
```
> r <- rbinom(n=500, size=1, p=.99)
> mean(r)
[1] 0.99
> var(r)
[1] 0.00991984
```

Theoretical values are .99 and .0099.

5.3.2 Binomial random variables

1.
```
> rbinom(p=0.15, size=25, n=24)
 [1] 5 4 5 5 4 2 3 3 1 2 4 2 3 2 5 5 4 5 3 3 3 8 5 4
> # Exceeds 5 once.
> rbinom(p=0.2, size=25, n=24)
 [1] 5 6 5 5 4 8 8 4 5 7 7 5 7 3 6 4 5 4 4 6 6 6 7 9
> # Exceeds 5 several times.
> rbinom(p=0.25, size=25, n=24)
 [1]  6  3  1 13  5 11 11  3  4  7  9  4  4  1  5  6  6  7  9  6  9  7  7  9
```

3.
```
> r <- rbinom(n=1000, size=18, p=.76)
> mean(r)
[1] 13.697
> var(r)
[1] 3.472664
```

Theoretical values are 13.68 and 3.2832.

5. (a)
```
> #Generate binomial pseudorandom variables using inversion method
> ranbin2 <- function(n, size, prob){
+         #'singlenumber' generates one binomial random variable.
+         singlenumber <- function(size, prob){
+         x <- runif(size)
+         N <- sum(x<prob)
+         N
+     }
+     replicate(n, singlenumber(size, prob))
+ }
```
   (b)
```
> system.time(gcFirst=T, ranbin(n=10000, size=10, prob=.4))
```

```
   user  system elapsed
   0.15    0.00    0.16
> system.time(gcFirst=T, rbinom(n=10000, size=10, prob=.4))
   user  system elapsed
   0.02    0.00    0.02
> system.time(gcFirst=T, ranbin(n=10000, size=100, prob=.4))
   user  system elapsed
   0.18    0.00    0.19
> system.time(gcFirst=T, rbinom(n=10000, size=100, prob=.4))
   user  system elapsed
      0       0       0
> system.time(gcFirst=T, ranbin(n=10000, size=1000, prob=.4))
   user  system elapsed
   0.33    0.00    0.33
> system.time(gcFirst=T, rbinom(n=10000, size=1000, prob=.4))
   user  system elapsed
   0.01    0.00    0.02
```

5.3.3 Poisson random variables

1.
```
> rpois(n=15, lambda=2.8)
 [1] 1 2 4 3 3 4 2 5 4 2 4 1 2 2 1
```

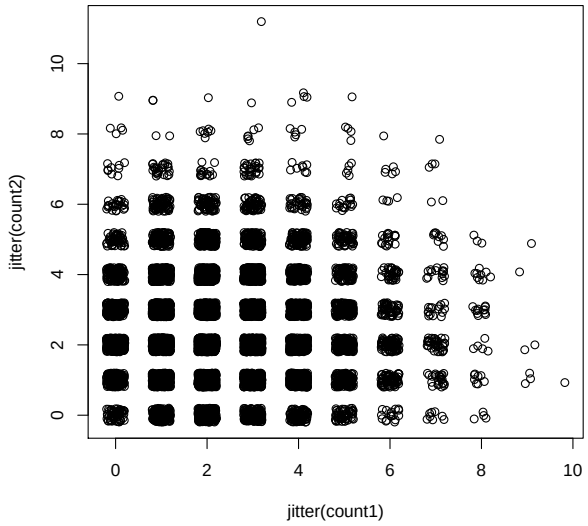
3.
```
> x <- rpois(10000, lambda=7.2)
> mean(x)
[1] 7.141
> var(x)
[1] 7.287248
```

7.
```
> N <- rpois(10000, lambda=2.5 * 2)
```

   (a)
```
> count1 <- c()
> count2 <- c()
> for(i in 1:10000){
+     u <- runif(N[i], max=2)
+     count1[i] <- sum(u<1)
+     count2[i] <- sum(u>1)
+ }
> sum(count1)
[1] 24886
> sum(count2)
[1] 24889
```

   (b) yes

   (c)
```
> plot(jitter(count1), jitter(count2))
```



5.3.4 Exponential random numbers

1. `> r <- rexp(50000, rate=3)`

   (a) `> sum(r<1)/length(r)`
   `[1] 0.95204`
   `> pexp(1, rate=3)`
   `[1] 0.950213`

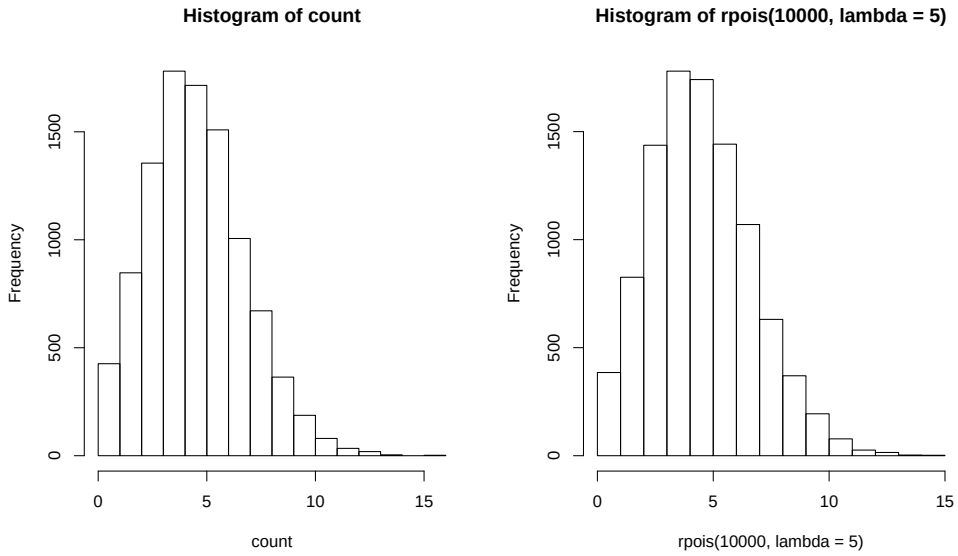
   (b) `> mean(r)`
   `[1] 0.3309469`

   (c) `> var(r)`
   `[1] 0.1083953`

3. ```
   > r1 <- rexp(100000, rate=1/3)
   > r2 <- rexp(100000, rate=1/6)
   > index <- (r2-r1)>0
   > r.min <- c(r1[index], r2[!index])
   > mean(r.min)
   [1] 1.984397
   > var(r.min)
   [1] 3.961419
   ```

5. ```
   > count <- c()
   > for(i in 1:10000){
   +     cum <- 0
   +     j <- 0
   +     while(cum<=2){
   +         j <- j+1
   +         r <- rexp(1, rate=2.5)
   +         cum <- cum + r
   +     }
   ```

```
+     count[i] <- j - 1
+ }
> par(mfrow=c(1,2))
> hist(count, breaks=20)
> hist(rpois(10000, lambda=5), breaks=20)
```



A QQ-plot would also be an appropriate way of checking this.

5.3.5 Normal random variables

1. ```
   > r <- rnorm(n=100, mean=51, sd=5.2)
   > mean(r)
   ```

   ```
   [1] 51.63488
   ```

   ```
   > sd(r)
   ```

   ```
   [1] 4.797961
   ```

3. ```
   > sim <- function(){
   +     ans <- 0
   +     while(abs(ans)<=2){
   +         ans <- rnorm(1, mean=3, sd=4)
   +     }
   +     ans
   + }
   > sim()
   ```

   ```
   [1] 7.188694
   ```

5. ```
   > r <- rchisq (n=100, df=8)
   > mean(r)
   ```

   ```
   [1] 8.36974
   ```

   ```
   > var(r)
   ```

   ```
   [1] 18.08655
   ```
   ```

5.4 Monte Carlo Integration

1. ```
   > #Monte Carlo
   > u <- runif(1000)
   > mean(u)

   [1] 0.5140157

   > #Integrate function
   > f <- function(x){
   +     x
   + }
   > integrate(f, lower=0, upper=1)

   0.5 with absolute error < 5.6e-15

   > #Monte Carlo
   > u <- runif(1000, max=3)
   > mean(u^2)*(3-0)

   [1] 9.138817

   > #Integrate function
   > f <- function(x){
   +     x^2
   + }
   > integrate(f, lower=0, upper=3)

   9 with absolute error < 1e-13

   > #Monte Carlo
   > u <- runif(1000, max=pi)
   > mean(sin(u))*(pi-0)

   [1] 1.988680

   > #Integrate function
   > f <- function(x){
   +     sin(x)
   + }
   > integrate(f, lower=0, upper=pi)

   2 with absolute error < 2.2e-14

   > #Monte Carlo
   > u <- runif(1000, min=1, max=pi)
   > mean(exp(u)/dunif(u,min=1, max=pi))

   [1] 20.51446

   > #Integrate function
   > f <- function(x){
   +     exp(x)
   + }
   > integrate(f, lower=1, upper=pi)

   20.42241 with absolute error < 2.3e-13

   > #Monte Carlo
   > r <- rexp(1000)
   > mean(exp(-r)/dexp(r))

   [1] 1
   ```
   ```

```
> #Integrate function
> f <- function(x){
+     exp(-x)
+ }
> integrate(f, lower=0, upper=1000)

1 with absolute error < 9e-10

> #Monte Carlo
> r <- rexp(1000)
> mean(exp(-r^3)/dexp(r))

[1] 0.9187494

> #Integrate function
> f <- function(x){
+     exp(-x^3)
+ }
> integrate(f, lower=0, upper=1000)

0.8929795 with absolute error < 8.4e-10

> #Monte Carlo
> r <- runif(1000, max=3)
> mean(sin(exp(r))/dunif(r, max=3))

[1] 0.617729

> #Integrate function
> f <- function(x){
+     sin(exp(x))
+ }
> integrate(f, lower=0, upper=3)

0.6061245 with absolute error < 2.8e-10

> #Monte Carlo
> r <- runif(1000,max=1)
> mean(dnorm(r))

[1] 0.3382709

> #Integrate function
> f <- function(x){
+     dnorm(x)
+ }
> integrate(f, lower=0, upper=1)

0.3413447 with absolute error < 3.8e-15

> #Monte Carlo
> r <- runif(1000,max=2)
> mean(dnorm(r)*2)

[1] 0.4830885

> #Integrate function
> f <- function(x){
+     dnorm(x)
+ }
> integrate(f, lower=0, upper=2)

0.4772499 with absolute error < 5.3e-15
```

```
> #Integrate function
> f <- function(x){
+     dnorm(x)
+ }
> integrate(f, lower=0, upper=3)

0.4986501 with absolute error < 5.5e-15
```

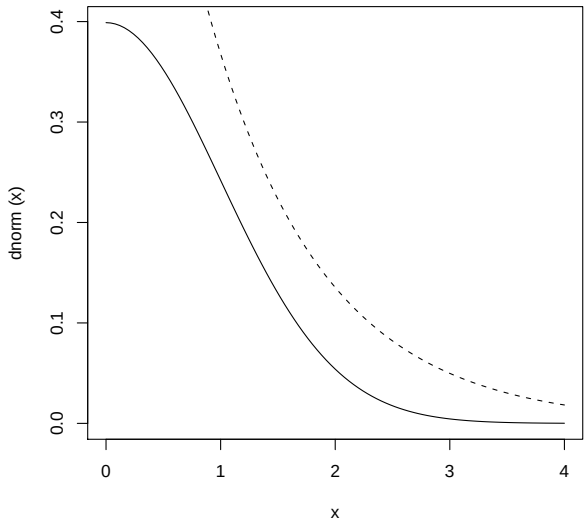
5.5 Advanced Simulation Methods

5.5.2 Importance Sampling

1. ```
   > rand.normal <- function(n){
   +     r1 <- runif(n, min=-4, max=4)
   +     r2 <- runif(n)
   +     ans <- r1[ r2 < 2.5*dnorm(r1)]
   +     while(length(ans) < n){
   +         r1 <- runif(n, min=-4, max=4)
   +         r2 <- runif(n)
   +         ans <- c(ans,r1[ r2 < 2.5*dnorm(r1)])
   +     }
   +     ans[1:n]
   + }
   ```

3. ```
   > r <- rannorm(10000,8,2)
   ```

5. ```
   > curve(dnorm, from =0, to =4)
   > curve(dexp, from =0, to =4, add=T, lty=2)
   ```




7. ```
   > set.seed(91626)
   > probs <- runif(100)
   > probs <- probs/sum(probs)
   > system.time(gcFirst=T, r <- randiscrete1(100, probs))

      user  system elapsed
         0       0       0
   ```

```
> system.time(gcFirst=T, r <- randiscrete1(1000, probs))

   user  system elapsed
   0.02    0.00    0.02

> system.time(gcFirst=T, r <- randiscrete1(10000, probs))

   user  system elapsed
    0.2     0.0     0.2

> system.time(gcFirst=T, r <- randiscrete2(100, probs))

   user  system elapsed
      0       0       0

> system.time(gcFirst=T, r <- randiscrete2(1000, probs))

   user  system elapsed
   0.06    0.00    0.06

> system.time(gcFirst=T, r <- randiscrete2(10000, probs))

   user  system elapsed
   0.69    0.00    0.69
```

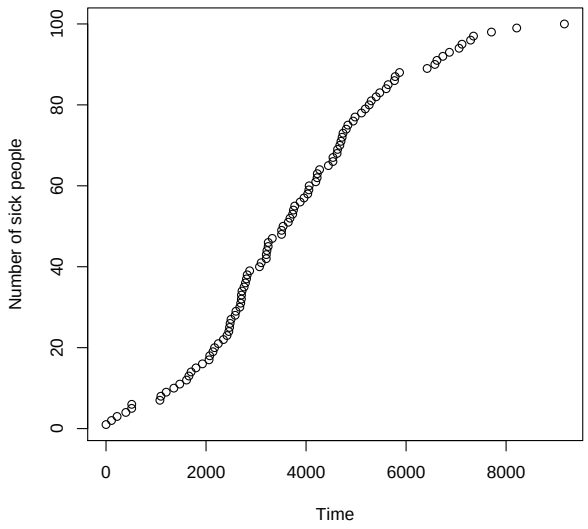
When inversion method is not available.

5.6 Chapter Exercises

1.
```
> simulate <- function(AnnFirst = T, pAnn, pBob){
+     scoreAnn <- 0
+     scoreBob <- 0
+     nextPlayer <- 'Ann'
+     if(AnnFirst==F){
+         nextPlayer <- Bob
+     }
+     while(scoreAnn < 21 & scoreBob < 21){
+         if(nextPlayer=='Ann'){
+             r <- rbinom(1, 1, pAnn)
+             if(r==1){
+                 scoreAnn <- scoreAnn + 1
+             }else{
+                 nextPlayer <- 'Bob'
+             }
+         }else{
+             r <- rbinom(1, 1, pBob)
+             if(r==1){
+                 scoreBob <- scoreBob + 1
+             }else{
+                 nextPlayer <- 'Ann'
+             }
+         }
+     }
+     return(nextPlayer)
+ }
> pAnn <- .7
> pBob <- .72
> result <- c()
> for(i in 1:7){
```

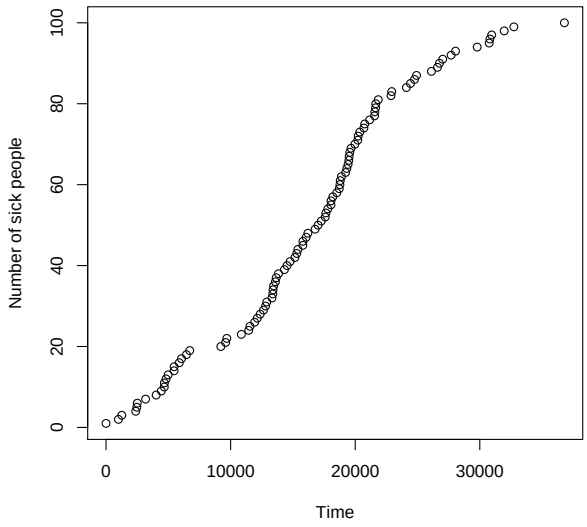
```
+     result <- c(result,simulate(AnnFirst=T, pAnn=pAnn, pBob=pBob))
+ }
```

2.
```
> simulate <- function(N, p){
+     time.line <- 0
+     time.current <- 0
+     n.sick <- 1
+
+     while(n.sick<N){
+         time.current <- time.current + 1
+
+         #if encounters==0, then none of the two persons is sick
+         #if encounters==1, then one of the two persons is sick
+         #if encounters==2, then both of the two persons are sick
+         encounters <- rhyper(nn=1, m=n.sick, n=N-n.sick, k=2)
+
+         if(encounters==1){
+             contage <- rbinom(n=1, size=1, p)
+             if(contage==1){
+                 #record current time
+                 time.line <- c(time.line, time.current)
+                 n.sick <- n.sick + 1
+             }
+         }
+     }
+     time.line
+ }
> N <- 100
> p <- .05
> ans <- simulate(N, p)
> plot(ans, 1:N, xlab='Time', ylab='Number of sick people')
```



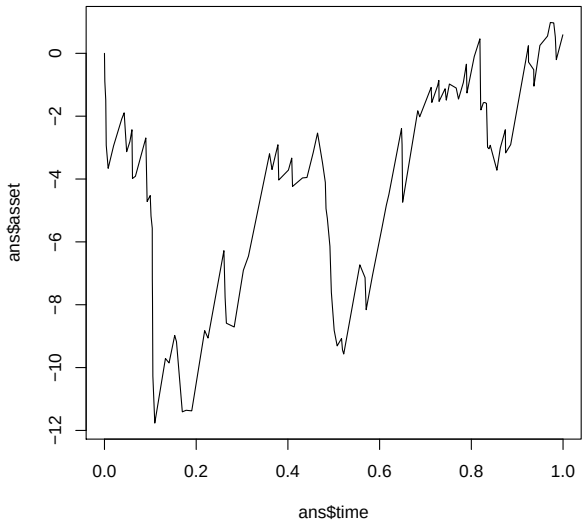
5.
```
> simulate <- function(N, p){
+     time.line <- 0
+     time.current <- 0
```

```
+     n.sick <- 1
+
+     while(n.sick<N){
+         time.current <- time.current + rexp(n=1,rate=1/5)
+
+         #if encounters==0, then none of the two persons is sick
+         #if encounters==1, then one of the two persons is sick
+         #if encounters==2, then both of the two persons are sick
+         encounters <- rhyper(nn=1, m=n.sick, n=N-n.sick, k=2)
+
+         if(encounters==1){
+             contage <- rbinom(n=1, size=1, p)
+             if(contage==1){
+                 #record current time
+                 time.line <- c(time.line, time.current)
+                 n.sick <- n.sick + 1
+             }
+         }
+     }
+     time.line
+ }
> N <- 100
> p <- .05
> ans <- simulate(N, p)
> plot(ans, 1:N, xlab='Time', ylab='Number of sick people')
```



7. (a)
```
> simulate <- function(){
+     N <- rpois(n=1, lambda=100)
+     claimSize <- rgamma(N, shape=2, rate=2)
+     claimTime <- sort(runif(N))
+
+     asset <- 0
+     asset.1 <- 105 * claimTime[1] - claimSize[1]
+     asset <- c(asset, asset.1)
```

```
+
+     for(i in 2:N){
+         len <- length(asset)
+         asset.i <- asset[len] + (claimTime[i] - claimTime[i-1])*105 - claimSize[i]
+         asset <- c(asset, asset.i)
+     }
+     claimTime <- c(0, claimTime)
+     list(time=claimTime, asset=asset)
+ }
> ans <- simulate()
> plot(ans$time, ans$asset, type='l')
```



(b)
```
> minimum <- c()
> final <- c()
> for(i in 1:1000){
+     ans <- simulate()
+     minimum <- c(minimum, min(ans$asset))
+     final <- c(final, rev(ans$asset)[1])
+ }
> mean(minimum)
[1] -6.723402
> mean(final)
[1] 4.721359
```