STAT 675 HW #1

Dominic D LaRoche

February 12, 2014

3.1 The pdf and cdf of the two parameter exponential distribution are:

$$f(x) = \lambda e^{-\lambda(x-\eta)}, F(x) = 1 - e^{-\lambda(x-\eta)}$$

Using the inverse transform method we find the inverse function $F^{-1}(u)$:

$$F^{-1}(u) = \eta + \frac{ln(1-u)}{-\lambda}$$

Then I can create a function to find a random sample of size n:

```
set.seed(36)
r.exp <- function(n, lambda, eta) {
    u <- runif(n)
    x <- (log(1 - u)/(-1 * lambda)) + eta
    return(x)
}
X <- r.exp(1000, 2, 3)
# to find theoretical quantiles
n = 1000
q <- qexp(ppoints(n), 2) + 3
```

I can then compare the sample to the theoretical quantiles as in figure 1.

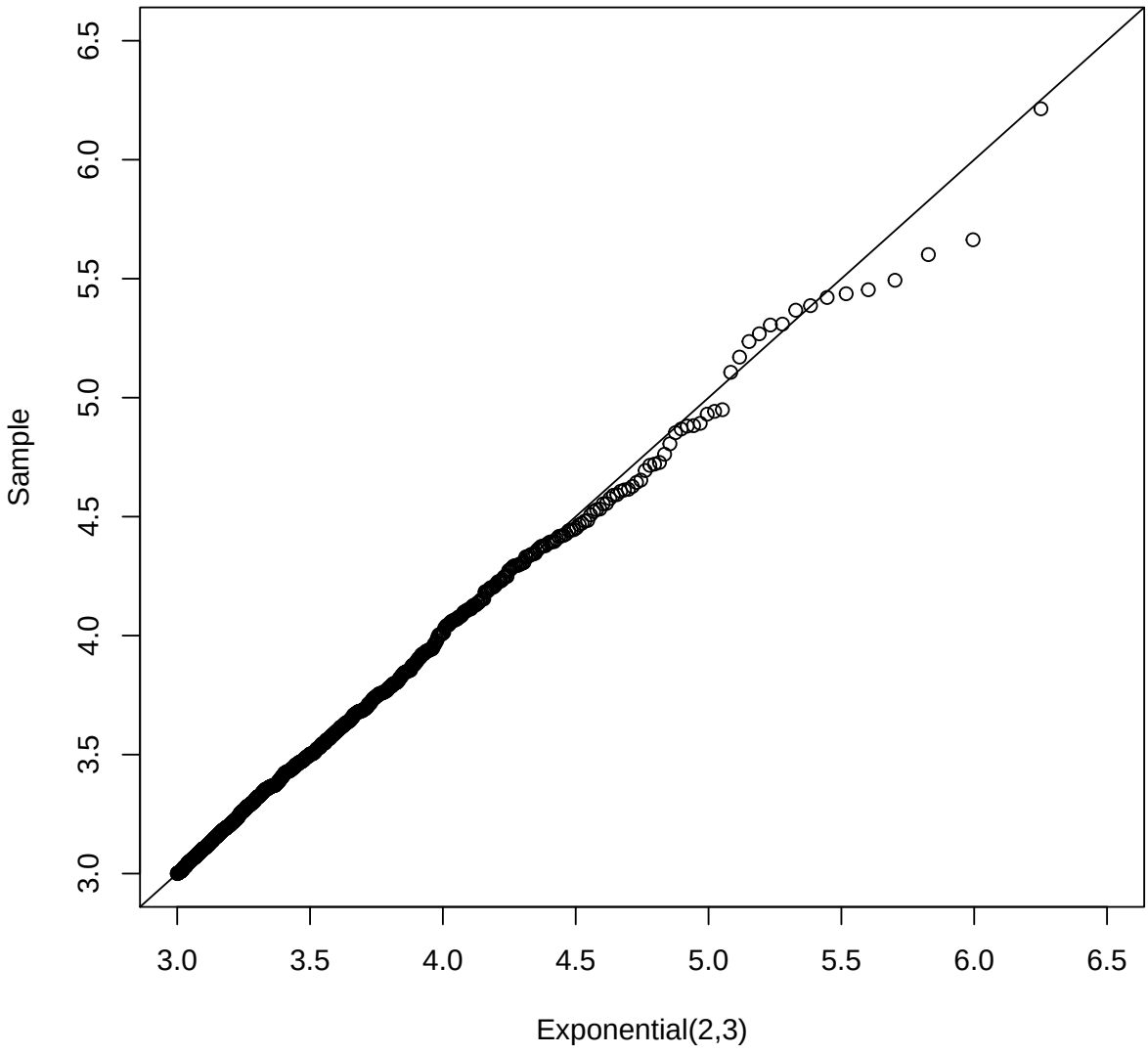

Figure 1: Comparison of the randomly generated and theoretical values of the exponential(2,3) distribution. The line represents perfect correspondence.

3.3 We first find the inverse of the Pareto cdf:

$$F^{-1}(u) = exp\left[-\frac{ln(1-u)}{a} + ln(b)\right], u \geq \frac{1}{b} > 0, a > 0$$

We then use this to generate random sample from a random uniform sample.

```
require(VGAM)
r_pareto <- function(n, loc, shape) {
    y <- c()
    i <- 1
    while (length(y) < n) {
        U <- runif(1, min = (1/loc), max = (1))
        x <- exp((-1 * log(1 - U)/shape) + log(loc))
        if (x > loc) {
            y[i] <- x
            i <- i + 1
        }
    }
    return(y)
}
x <- r_pareto(1000, 2, 2)
ppdf <- function(x, a, b) {
    (a * (b^a))/(x^(a + 1))
}
```

The histogram of the Pareto sample with a density curve from the actual Pareto distribution is in figure 2.

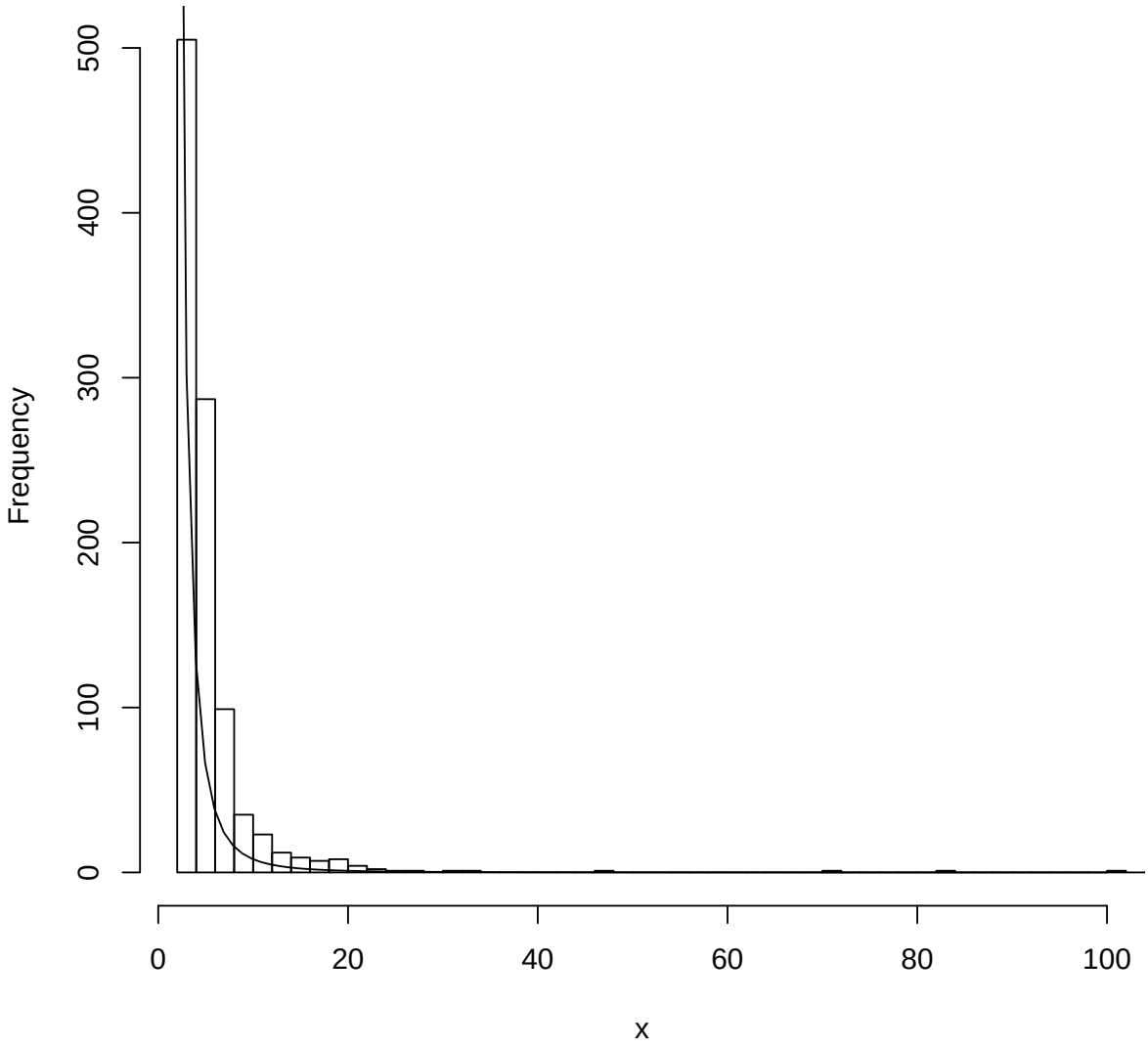

Figure 2: Sample from the Pareto generator with the theoretical density overlaid

3.4 Since I could not find a unique inverse to the Raleigh CDF I will use the accept/reject method. I will choose $g(y) \sim$ Exponential $(\beta = \sigma^2)$.

$$u < \frac{f(y)}{cg(y)} = \frac{\frac{x}{\sigma^2}e^{-x^2/2\sigma^2}}{\frac{c}{\sigma^2}e^{-x/\sigma^2}} = \frac{xexp(\frac{-x^2-2x}{\sigma^2})}{C}$$

The constant c will change depending on the value of σ chosen and therefore it will be useful to have a function which sets C based on σ. The maximum of the above function over the support $x \geq 0, \sigma > 0$ can be determined with some calculus to be:

$$x = \frac{\sqrt{1+4\sigma^2}}{2}$$

Several histograms with their associated theoretical modes and the number of random uniform variables required to produce a sample of 1000 are shown in figure 3. As can ebe seen in the above equation this generator becomes increasingly inefficient as σ grows.

```
r_ral <- function(n, sigma) {
    x <- c()
    i <- 1
    j <- 1
    s2 <- sigma^2
    c <- (1 + sqrt(1 + 4 * sigma^2))/2
    while (length(x) < n) {
        u <- runif(1)
        y <- rexp(1, rate = (1/(sigma^2)))
        j <- j + 1
        if (u < ((y * (exp((-1 * y^2 + 2 * y)/(2 * s2))))/c)) {
            x[i] <- y
            i <- i + 1
        }
    }
    return(list(x, j))
}
r_2 <- r_ral(1000, 2)
r_4 <- r_ral(1000, 4)
r_6 <- r_ral(1000, 6)
r_8 <- r_ral(1000, 8)
```

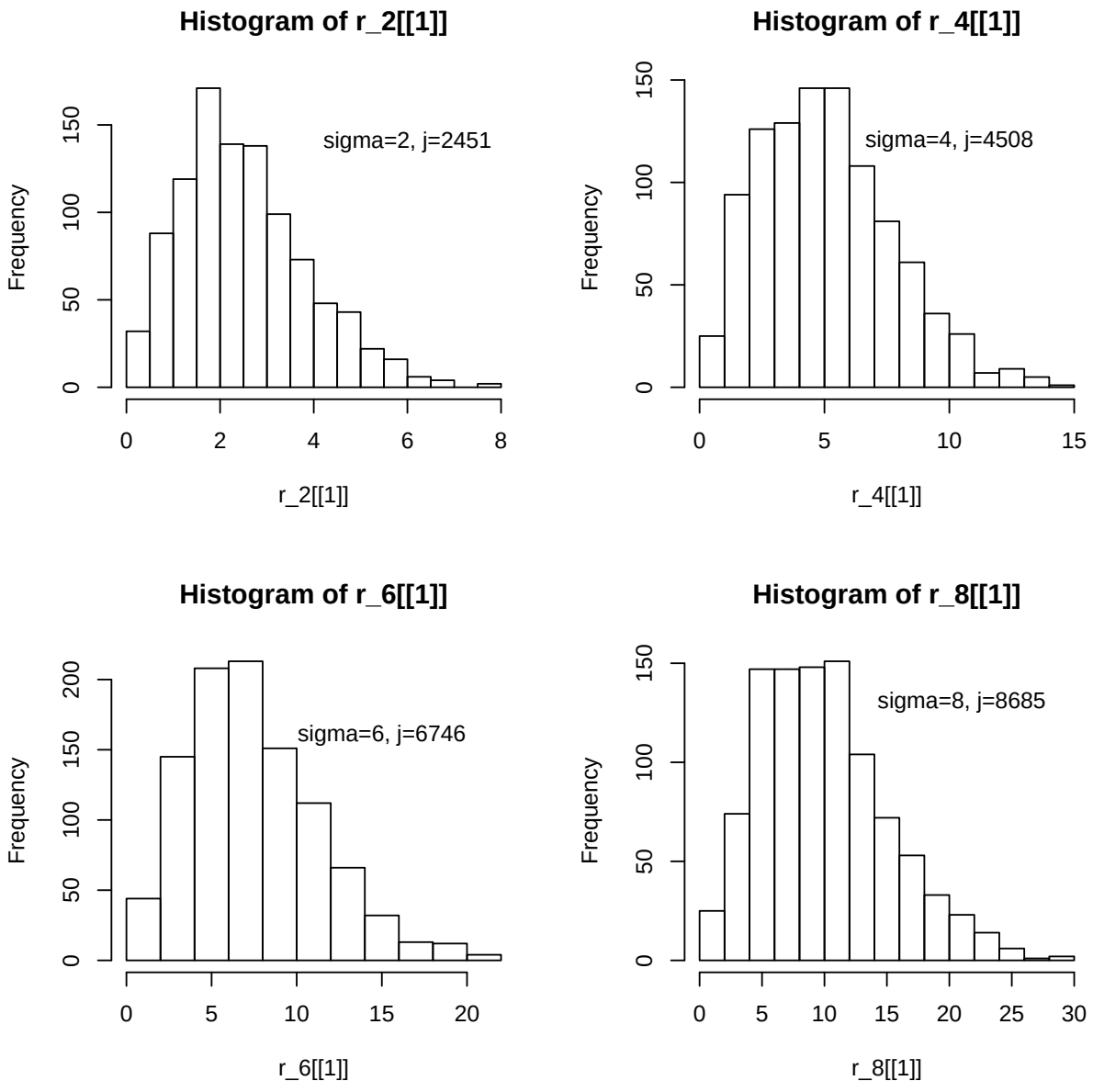

Figure 3: Several Raleigh distributions, the expected modes (sigma) and the number of random samples needed to generate 1000 numbers (j).

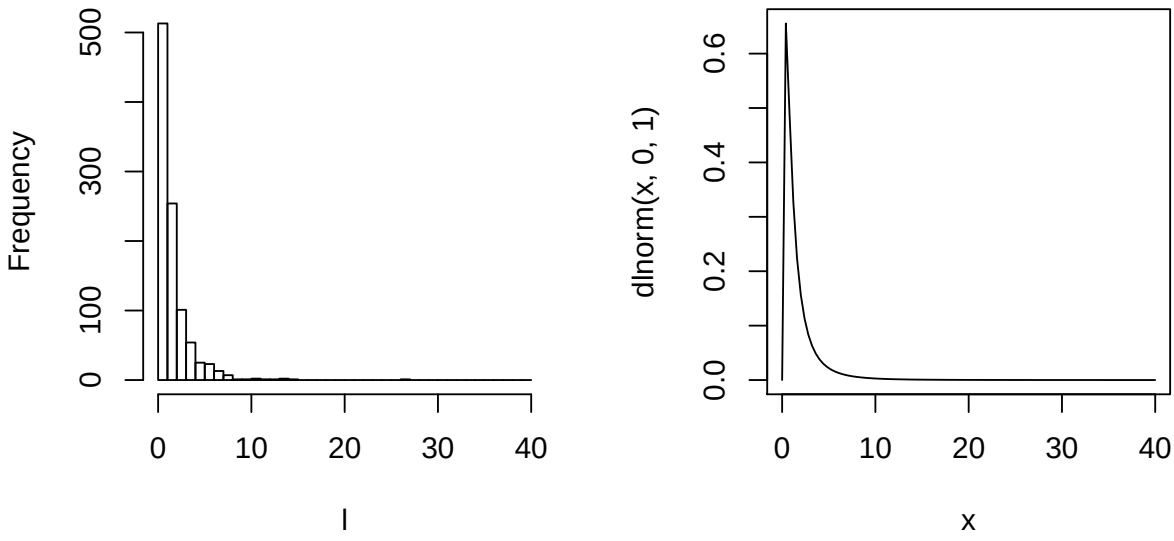

Figure 4: (
A histogram of the randomly generated lognormal sample and the theoretical density of the same distribution)
lnorm)

3.8 If Z $\sim$ Normal(μ, σ^2) then e^Z $\sim$Lognormal$(\mu = e^{\mu+(\sigma^2/2)}, \sigma^2 = e^{2(\mu+\sigma^2)} - e^{2\mu+\sigma^2})$. Therefore to get a lognormal(μ, σ^2) we will need to generate a Z$\sim$Normal and return e^Z. Figure **??** shows the random sample generated from a lognormal(0,1) distribution and the theoretical density.

```
r_lnorm <- function(n, mu, sigma2) {
    x <- rnorm(n, mu, sigma2)  #mu+sigma2/2,2*(mu+sigma2)+log(1-exp(-sigma2)))
    y <- exp(x)
    return(y)
}
l <- r_lnorm(1000, 0, 1)
```

3.9 The rescaled Epanechnikov distribution can be simulated with:

```
r_ekov <- function(n) {
    x <- c()
    i <- 1
    while (length(x) < n) {
        u1 <- runif(1, min = -1, max = 1)
        u2 <- runif(1, min = -1, max = 1)
        u3 <- runif(1, min = -1, max = 1)
        x[i] <- ifelse(u3 >= u2 && u3 >= u1, u2, u3)
        i = i + 1
```

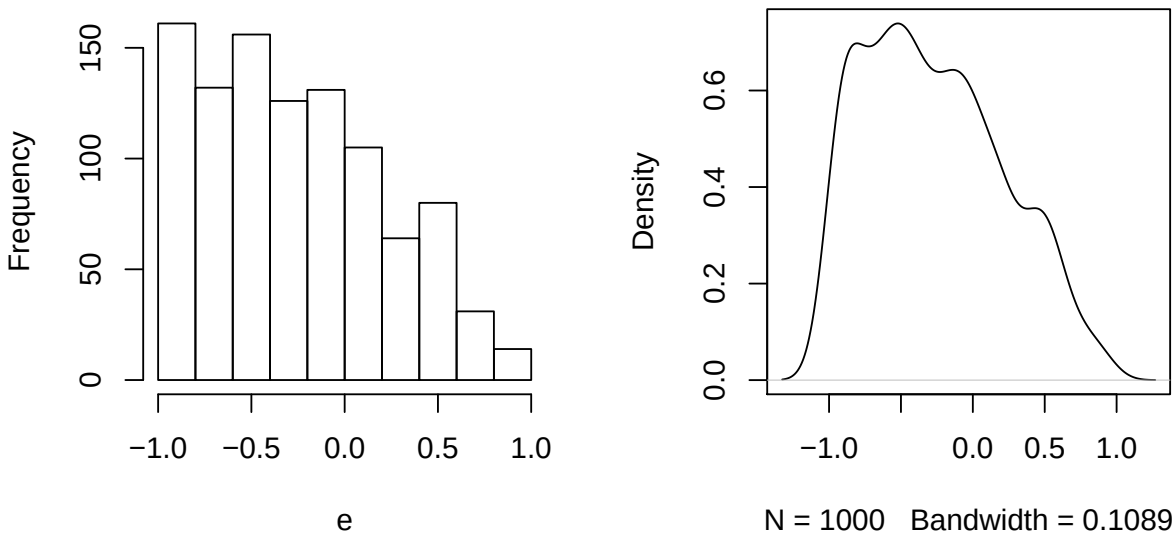

Figure 5: Histogram of the randomly generated Epanechnikov distribution and a corresponding density estimate

```
    }
    return(x)
}
e <- r_ekov(1000)
```

The histogram and density estimate are shown in figure 5.

3.11 Mixture of Normal(0,1) and Normal(3,1):

```
rlocmix <- function(n, l1, l2, p1) {
    loc <- sample(c(l1, l2), size = n, replace = TRUE, prob = c(p1, 1 - p1))
    x <- rnorm(n, loc, 1)
    return(x)
}
m <- rlocmix(1000, 0, 3, 0.75)
m2 <- rlocmix(1000, 0, 3, 0.5)
```

Based on the results of figure 6, and some simple logic, I would conjecture that p near 0 or 1 would produce a unimodal distriution with skew whereas a p near 0.5 would produce a bimodal distribution.

3.12 Continuous mixture of Exponential(Λ) with $\Lambda \sim$Gamma(r, β).

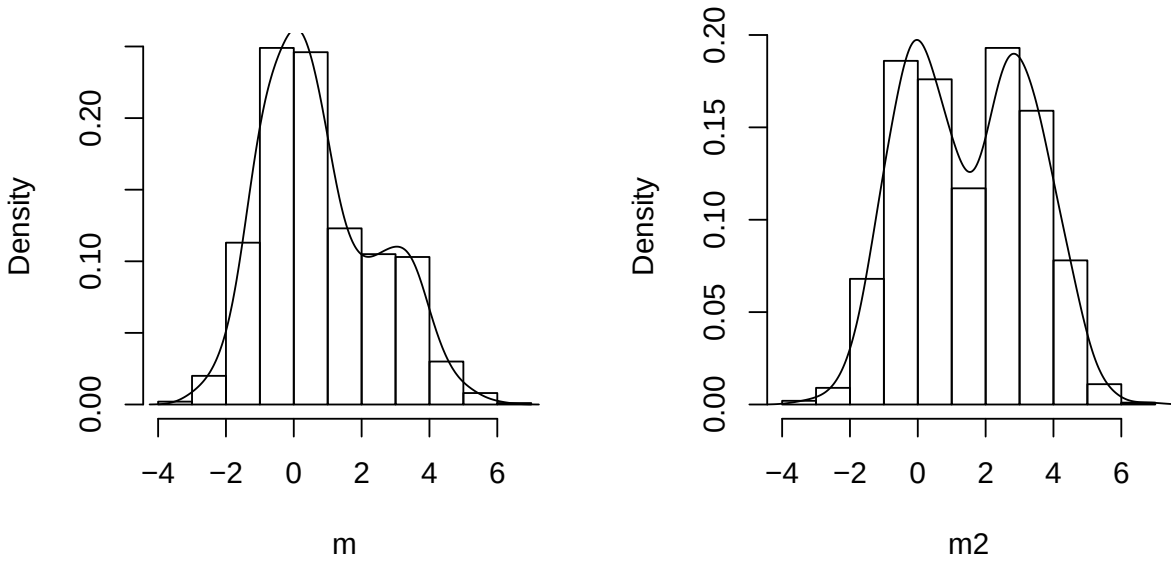

Figure 6: Two normal location mixtures one with p=.75 (left) and one with p=.5 (right). I would conjecture that p near 0 or 1 would produce a unimodal distriution with skew whereas a p near .5 would produce a bimodal distribution.

```
lam <- rgamma(1000, shape = 2, rate = 4)
x <- rexp(1000, rate = lam)
rm(list = ls())
```

3.14 This is taken right from example 3.18, results are in figure 7:

```
rmvn.Choleski <- function(n, mu, Sigma) {
    d <- length(mu)
    Q <- chol(Sigma)
    Z <- matrix(rnorm(n * d), nrow = n, ncol = d)
    X <- Z %*% Q + matrix(mu, n, d, byrow = TRUE)
    X
}

mu <- c(0, 1, 2)
Sigma <- matrix(c(1, -0.5, 0.5, -0.5, 1, -0.5, 0.5, -0.5, 1), 3, 3)
X <- rmvn.Choleski(200, mu, Sigma)
```

3.18 Simulate the Weishart distribution:

```
# generate lower triangular N(0,1) matrix T with d=5
T <- matrix(rnorm(15), 5, 5)
T[upper.tri(T)] <- 0
i <- 1:5
d <- sqrt(rchisq(length(i), 6 - i + 1))
diag(T) <- d
T  #lower triangular matrix of N(0,1)

##         [,1]    [,2]    [,3]   [,4]  [,5]
## [1,]  2.6822  0.0000  0.0000 0.0000 0.000
## [2,]  0.8588  1.8874  0.0000 0.0000 0.000
## [3,] -1.7585 -1.8231  1.2723 0.0000 0.000
## [4,] -0.4917  1.8010 -0.1232 1.1242 0.000
## [5,]  0.2302 -0.2214 -0.5310 0.2302 1.124

A <- T %*% t(T)
A

##         [,1]    [,2]    [,3]    [,4]    [,5]
## [1,]  7.1944  2.3036 -4.7169 -1.3189  0.6175
## [2,]  2.3036  4.2999 -4.9512  2.9770 -0.2202
## [3,] -4.7169 -4.9512  8.0348 -2.5755 -0.6768
## [4,] -1.3189  2.9770 -2.5755  4.7646 -0.1877
## [5,]  0.6175 -0.2202 -0.6768 -0.1877  1.6998

# make symmetric sigma matrix
require(Matrix)
```

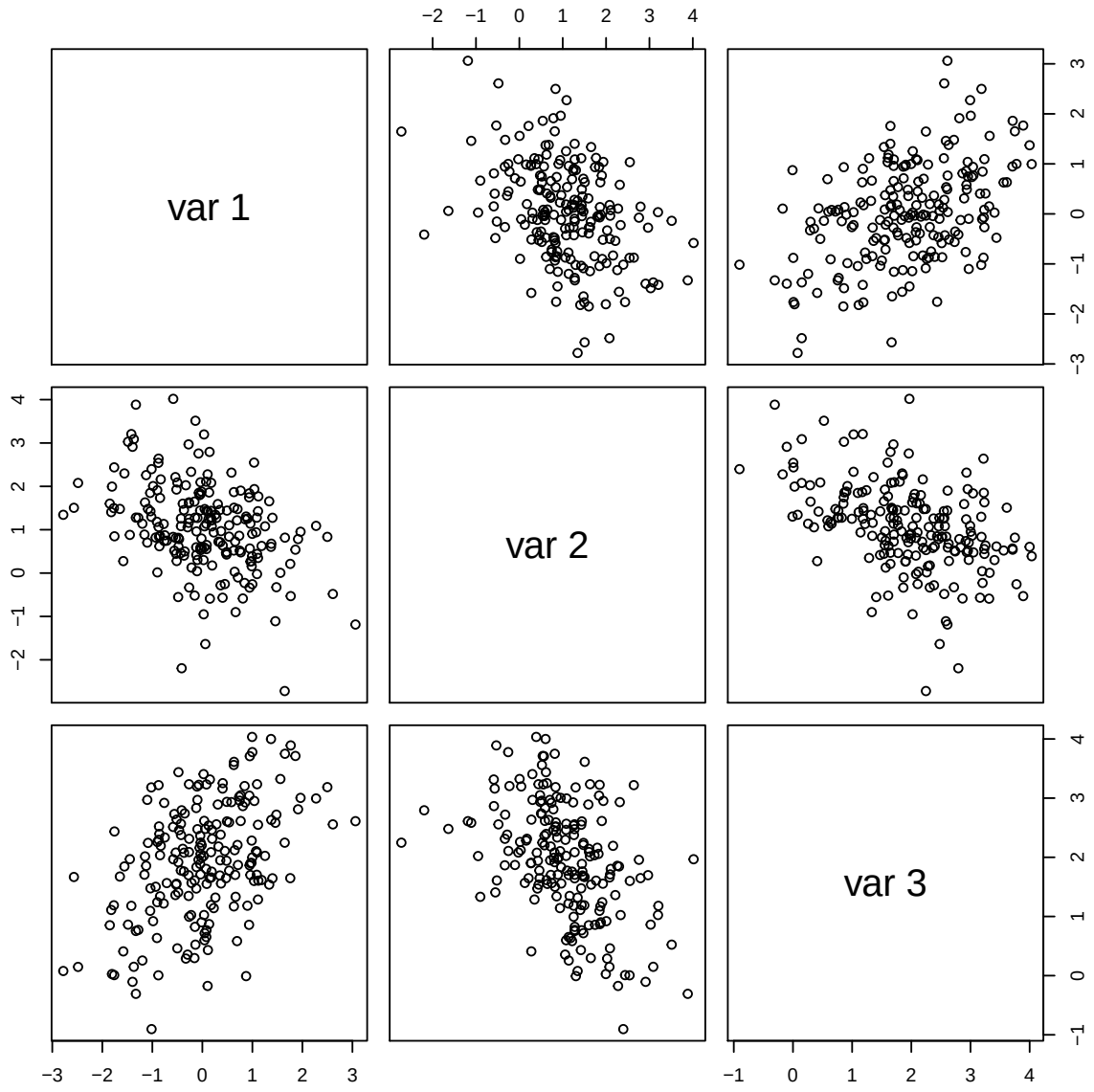

Figure 7: The scatterplot matrix from the multivariable normal sample relfect the covariance matrix (Σ) used to generate them.

```
sigma <- forceSymmetric(matrix(c(1, 0, 0, 0, 0, 0.5, 1, 0, 0, 0, 0.5, 0.5, 1,
    0, 0, 0.5, 0.5, 0.5, 1, 0, 0.5, 0.5, 0.5, 0.5, 1), 5, 5))
l <- chol(sigma)
x <- t(l) %*% A %*% l
x

## 5 x 5 Matrix of class "dgeMatrix"
##        [,1]   [,2]    [,3]   [,4]    [,5]
## [1,] 7.1944  5.592  0.4109  2.257  3.5691
## [2,] 5.5922  7.018 -1.2231  4.364  3.2417
## [3,] 0.4109 -1.223  1.9931 -1.209 -0.2438
## [4,] 2.2567  4.364 -1.2092  4.073  1.8648
## [5,] 3.5691  3.242 -0.2438  1.865  2.7662

rm(list = ls())
```

3.19 Random symmetric walk, figure 8 shows the results:

```
A <- 10
trace <- vector()
i <- 1
while (A < 20 && A > 0) {
    win <- sample(c(-1, 1), size = 1, prob = c(0.5, 0.5))
    trace[i] <- A  #place the trace before updating A to get A(0)
    A <- A + win
    i <- i + 1
}
A

## [1] 0

trace

##   [1] 10 11 12 11 10 11 10 11 10 11 12 13 14 13 14 15 16 15 16 15 14 15 14
##  [24] 13 14 13 12 11 12 11 12 11 12 11 12 13 14 15 16 17 18 17 16 17 18 17
##  [47] 18 17 18 17 18 19 18 17 18 17 18 17 16 15 16 17 16 17 16 15 14 15 16
##  [70] 15 14 13 12 11 12 11 10 11 12 11 12 11 10 11 10 11 10 11 10 11 10 11
##  [93] 12 11 10  9  8  7  6  5  4  5  4  5  6  7  6  7  8  7  8  9 10 11 12
## [116] 11 12 13 12 13 12 11 10  9  8  9  8  7  8  7  8  7  6  5  6  7  8  9
## [139]  8  7  6  7  8  7  8  9  8  7  8  9 10  9 10 11 10  9  8  9  8  7  8
## [162]  7  6  7  8  7  6  7  6  5  6  7  8  7  6  5  6  5  6  7  8  9 10  9
## [185]  8  7  6  5  4  5  6  5  6  5  6  5  6  7  6  7  6  7  8  7  8  9 10
## [208]  9 10 11 12 11 10  9  8  9  8  9  8  9 10 11 10  9  8  7  6  5  4  3
## [231]  4  5  6  5  4  3  2  3  4  3  4  5  6  7  8  9 10 11 12 11 12 13 14
## [254] 15 14 15 14 15 14 13 12 13 12 11 10  9 10  9 10 11 10  9  8  7  8  7
## [277]  8  7  8  7  8  7  6  5  4  5  4  5  6  5  4  5  4  5  4  5  6  5  6
## [300]  5  6  7  6  7  8  7  8  7  8  7  8  9  8  7  6  5  4  3  2  3  2  1

t <- 1:length(trace)
```

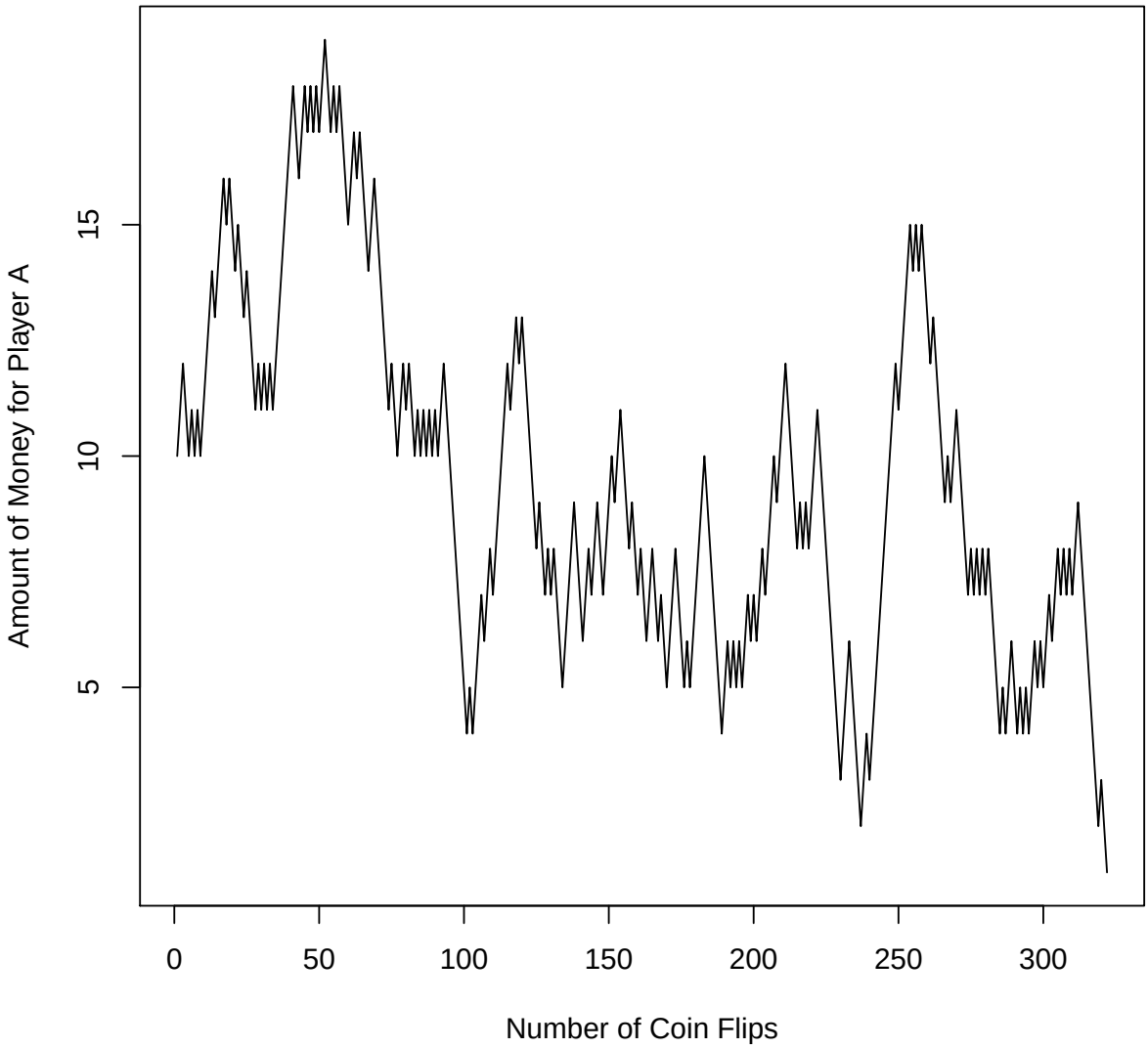

Figure 8: Plot of the random symmetric walk.