Homework 6

Zachary Lazerick

14 March 2023

1) The Multivariate Normal Distribution – Generate random variables X_1, X_2, and X_3 having a multivariate normal distribution with mean vector $\mu = (1, 2, 3)$ and covariance matrix

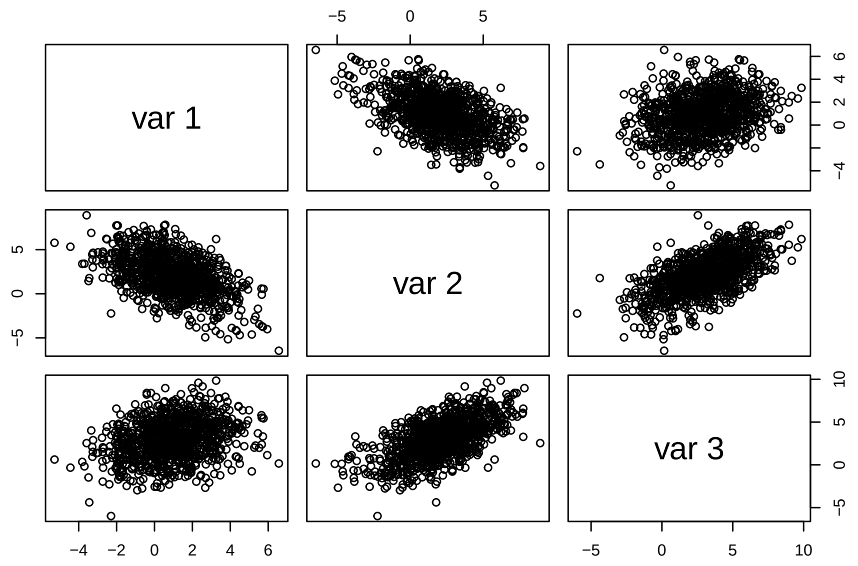
$$\begin{bmatrix} 3 & -2 & 1 \\ -2 & 5 & 3 \\ 1 & 3 & 5 \end{bmatrix}$$

Save your values in a $nx3$ matrix X and then use the R function pairs(X) plot to graph an array of scatterplots for each pair of variables. Visually check that the location and correlation (see note #1 below) approximately agree with the theoretical parameters of the corresponding bivariate normal distribution. Notes:

- $corr(X_1, X_2) = \frac{cov(X_1,X_2)}{\sigma_1 \sigma_2}$
- Remember that you need to use '%*%' to perform matrix multiplication

```r
MultivariateNormal <- function(iterations, mu, Sigma) {
  X <- matrix(data = NA, nrow = iterations, ncol = 3, byrow = T)
  A <- chol(Sigma)
  for (i in 1:iterations) {
    Z <- rnorm(3)
    X[i, 1:3] <- t(Z%*%A + mu)
  }
  return(X)
}

set.seed(1838)
mu <- c(1, 2, 3); Sigma <- matrix(c(3, -2, 1, -2, 5, 3, 1, 3, 5), nrow = 3, ncol = 3, byrow = T)
results1 <- MultivariateNormal(1000, mu, Sigma)
```

```r
## Visually represent the Variance/Covariance Matrix
pairs(results1)
```



```
## Check the Simulated Mean Vector Mu (X_1, X_2, X_3)
colMeans(results1)
```

```
## [1] 0.9494679 2.0507314 2.9588285
```

```
## Check the Simulated Correlations for each Pair of Variables
cor(results1[, 1], results1[, 2]); cor(results1[, 1], results1[, 3]); cor(results1[, 2], results1[, 3])
```

```
## [1] -0.4982155
```

```
## [1] 0.2683577
```

```
## [1] 0.6128331
```

```
## Check the Simulated Variance Vector Sigma^2 (X_1, X_2, X_3)
cov(results1)
```

```
##            [,1]      [,2]     [,3]
## [1,]  3.102092 -1.999082 1.089334
## [2,] -1.999082  5.190057 3.217719
## [3,]  1.089334  3.217719 5.311773
```

All simulated values for the mean and variance are within an acceptable margin of error for the true given values.

2) Standardizing a Multivariate Normal. Write a function that will standardize a multivariate normal sample for arbitrary sample size n and dimension d. That is, transform a sample so that the sample mean vector is zero and sample covariance is the identity matrix. To check your results, generate multivariate normal samples using the values of μ and Σ from question #1 and print the sample mean vector and covariance matrix before and after standardization. Notes: colMeans() calculates the mean of each column of a matrix, cov() creates a covariance matrix, and solve() inverts a matrix.

```r
StandardizeNormal <- function(matrix) {
  ## Find and Invert and Upper-Triangular Matrix A such that Sigma = A^T*A
  A <- chol(cov(matrix)); A.inverse <- solve(A)
  ## Compute Vector of Means
  Mu <- colMeans(matrix)

  Z <- matrix(data = NA, nrow = nrow(matrix), ncol = ncol(matrix), byrow = T)
  ## Compute Z
  for (i in 1:nrow(matrix)) {
    Z[i, 1:ncol(matrix)] <- (matrix[i, 1:ncol(matrix)] - Mu)%*%A.inverse
  }
  return(Z)
}

results2 <- StandardizeNormal(results1)

## Check Mu is the 0 Vector
colMeans(results2)
```

```
## [1]  5.664913e-17 -5.734302e-17  1.154493e-16
```

```r
## Check Sigma is the Identity Matrix
cov(results2)
```

```
##               [,1]          [,2]          [,3]
## [1,]  1.000000e+00  1.440855e-16 -3.286639e-16
## [2,]  1.440855e-16  1.000000e+00 -4.126049e-16
## [3,] -3.286639e-16 -4.126049e-16  1.000000e+00
```

The simulated standardized values for the matrix are within an acceptable margin of error for the zero vector for the mean vector μ, and the identity matrix for the covariance matrix Σ.

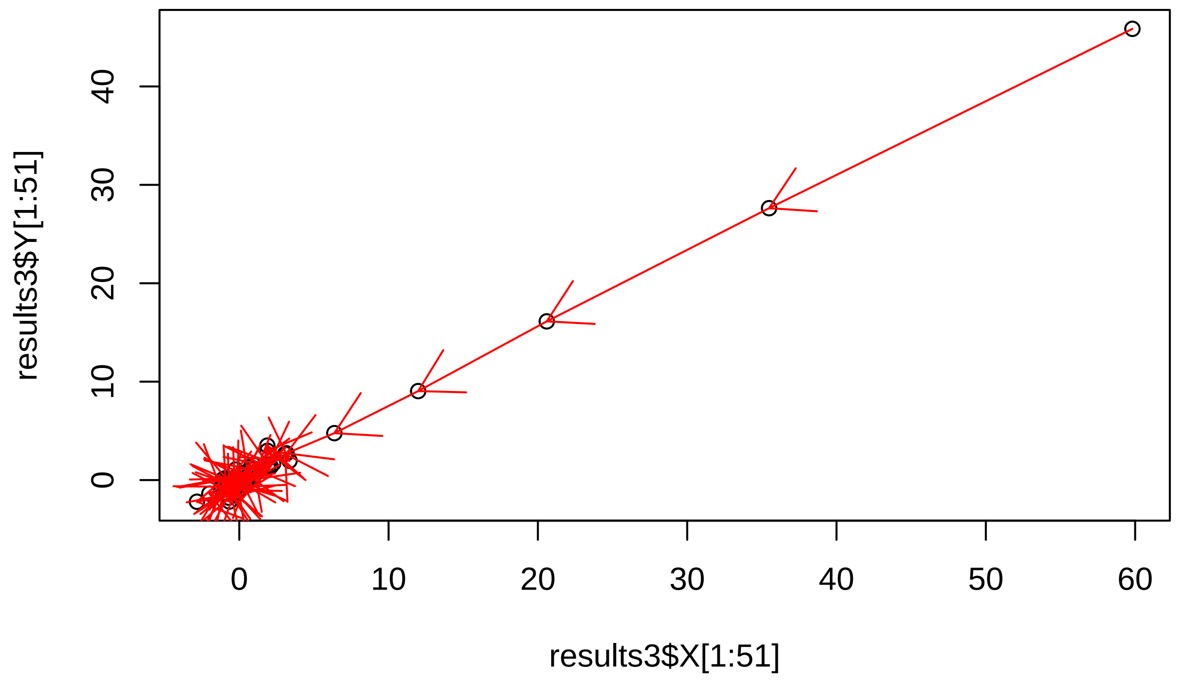
3) Recognizing Burn-In and Autocorrelation. Let X and Y have a bivariate normal density f(x,y) given by:

$$f(x,y) = \frac{1}{2\pi(1-\rho^2)} exp(-\frac{x^2+y^2-2\rho xy}{2(1-\rho^2)})$$

[so X and Y are standard normal random variables with correlation coefficient ρ]. Write a function that uses Gibbs sampling to produce realizations from this joint density, where $\rho = 0.75$. Your function should be flexible enough to accommodate a choice of (i) x_0 and y_0, (ii) Number of desired (x_i, y_i) realizations, (iii) Number of iterations to skip (lag) in order to avoid serial correlation. Use this program to complete the following:

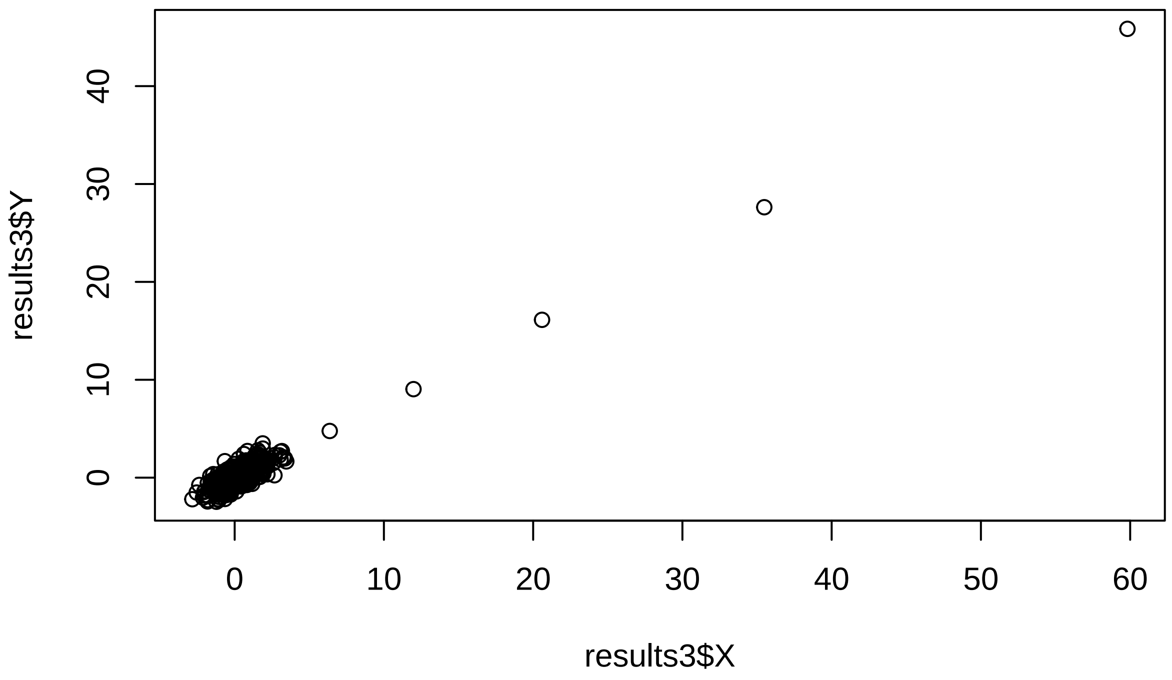
a) Saving every iteration, simulate 500 realizations from starting values $x_0 = 80$ and $y_0 = 80$. Plot the first 50 realizations in R, and connect the points in the order you generated them using the "arrows" command [see https://stat.ethz.ch/R-manual/R-devel/library/graphics/html/arrows.html] You may need to install the 'graphics' package before using "arrows". This function will draw arrows between the points instead of a straight line, which should help you follow the path the algorithm takes. Syntax: plot(X[1:51],Y[1:51]), arrows(x_0 = X[1:50], y_0 = Y[1:50], x_1 = X[2:51], y_1 = Y[2:51], col='red').

```r
BivariateBurnIn <- function(iterations, initial_x, initial_y, rho, lag = 1) {
  X.cord <- c(); Y.cord <- c(); counter <- 0
  next_x <- initial_x; next_y <- initial_y
  while (length(X.cord) < iterations) {
    next_x = rnorm(1, mean = rho*next_y, sd = sqrt(1-rho^2))
    next_y = rnorm(1, mean = rho*next_x, sd = sqrt(1-rho^2))
    if (counter %% lag == 0) {
      X.cord <- append(X.cord, next_x); Y.cord <- append(Y.cord, next_y)
      counter <- counter + 1
    }
    else {
      counter <- counter + 1
    }
  }
  mylist <- list("X" = X.cord, "Y" = Y.cord)
  return(mylist)
}

results3 <- BivariateBurnIn(500, 80, 80, .75)

plot(results3$X[1:51], results3$Y[1:51])
arrows(x0 = results3$X[1:50], y0 = results3$Y[1:50],
       x1 = results3$X[2:51], y1 = results3$Y[2:51], col='red')
```



b) Construct the same plot as in part (a) (but without the arrows), only this time use all 500 realizations. This type of plot gives an idea of the necessary 'burn-in' iterations for a given starting value.

```
plot(results3$X, results3$Y)
```

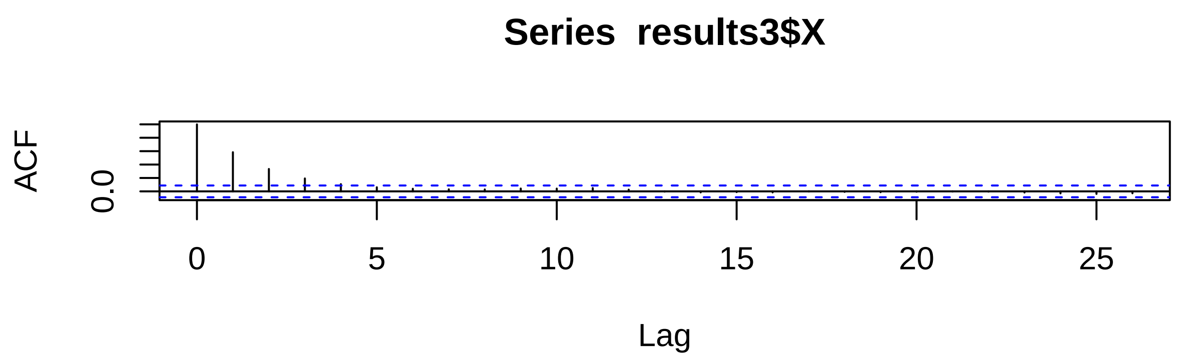


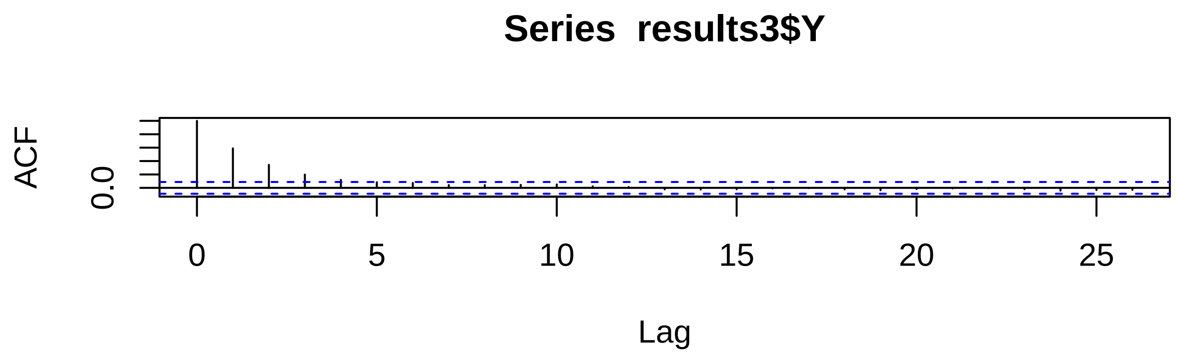
c) When using $x_0 = 80$ and $y_0 = 80$ as initial values, how many iterations (roughly) does it take for the Gibbs sampler to converge to its stationary distribution? This is where the arrows might come in handy.

It takes about 6 iterations for the Gibbs Sampler to converge to its stationary distribution.

d) Now create two autocorrelation plots - one for the 500 x-values and one for the 500 y-values. Based on these plots, what lag should be used to eliminate autocorrelation? [Use the R function acf(). The blue line on the graph indicates a significant correlation at a particular lag. Once you drop below the blue line, this indicates how many consecutive observations you want to consider for autocorrelation.]

```
layout(1:2)
acf(results3$X); acf(results3$Y)
```

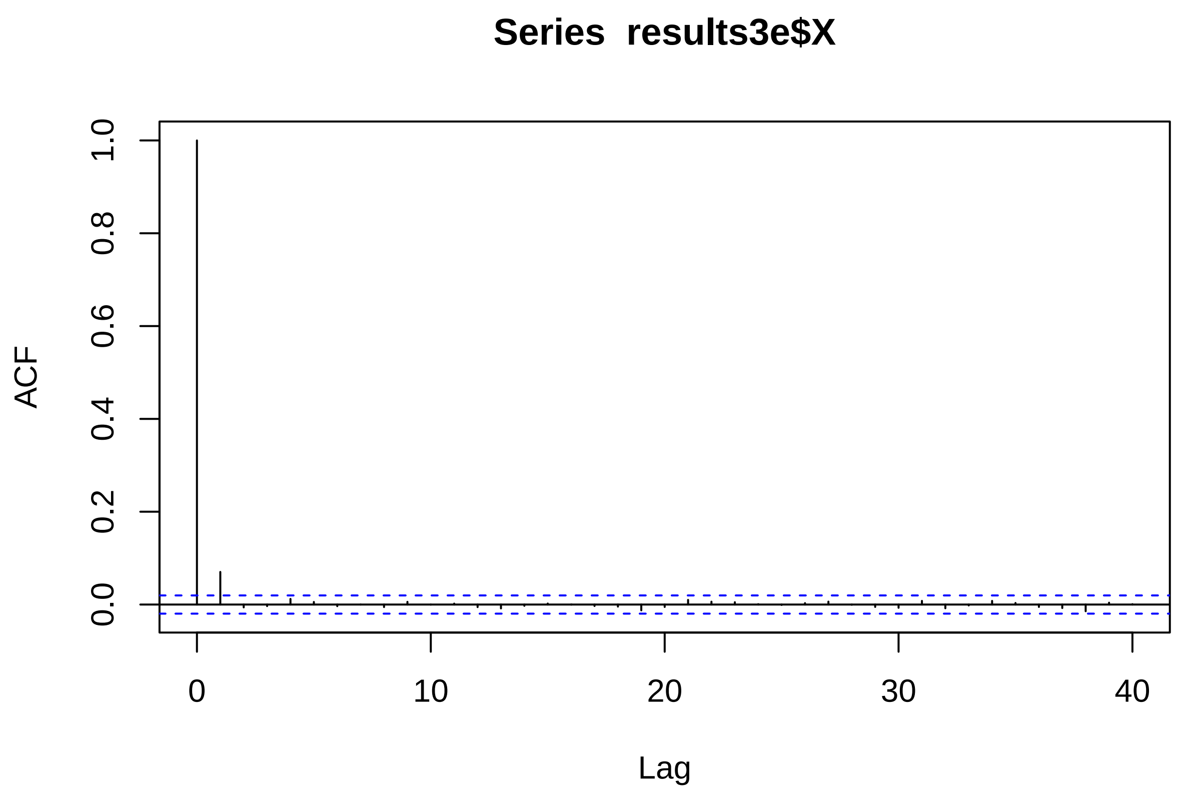




The autocorrelation is significant until lag 5. So, we should use every 5th observation to avoid serial correlation between points.

e) Using the lag from your answer to part (d), use your Gibbs sampling program to draw 10000 independent realizations from the marginal density of X. If done correctly, the acf() function should no longer show any correlation between the simulated values [check this]. Confirm, using mean, variance, and quartiles, that these realizations are indeed from a standard normal density. Syntax: quantile(X, probs = seq(0, 1, 0.1)), qnorm(seq(0, 1, 0.1)).

```
set.seed(520)
results3e <- BivariateBurnIn(10000, 80, 80, .75, lag = 5)

## Check Autocorrelation of Simulated X
acf(results3e$X)
```



```
## Check Mean, Variance of Simulated X
mean(results3e$X); var(results3e$X)
```

```
## [1] -0.0003765139
```

```
## [1] 1.347348
```

```
## Check Q-Q Plot
quantile(results3e$X, probs = (seq(0, 1, .1)))
```

```
##          0%         10%         20%         30%         40%         50% 
## -3.87054352 -1.28129340 -0.85239293 -0.53766384 -0.26720749 -0.01068243 
##         60%         70%         80%         90%        100% 
##  0.24881154  0.52624371  0.84682018  1.29239759 59.28865331
```

```
qnorm(seq(0, 1, .1))
```

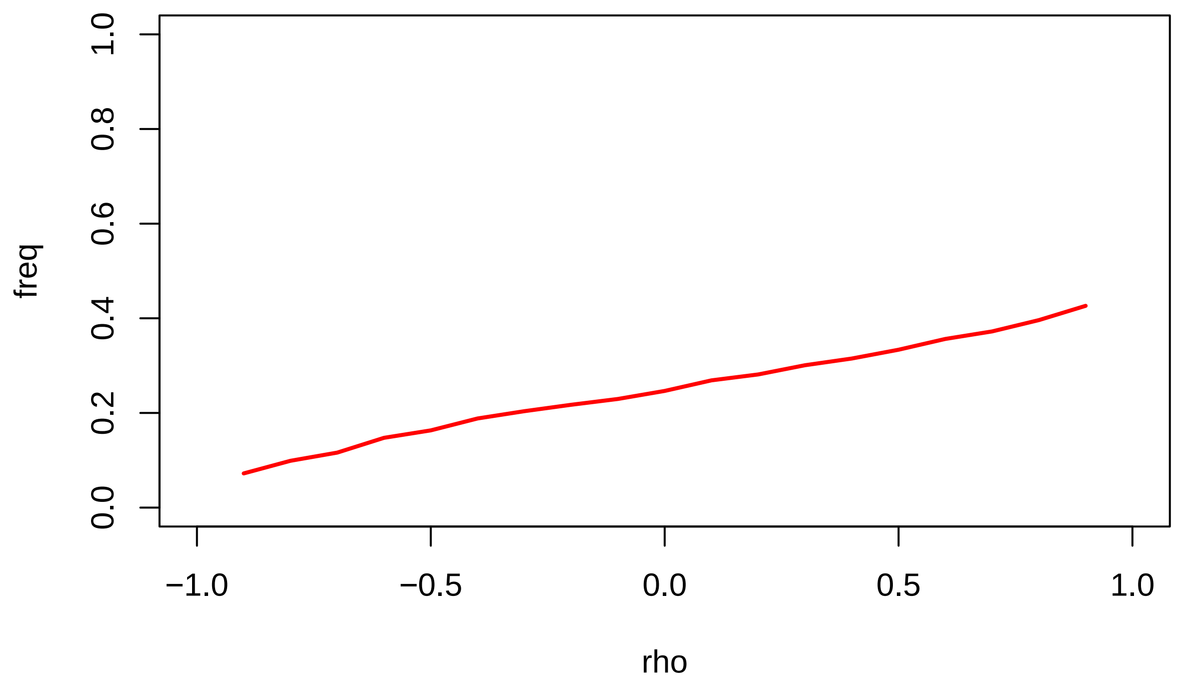
```
##  [1]       -Inf -1.2815516 -0.8416212 -0.5244005 -0.2533471  0.0000000
##  [7]  0.2533471  0.5244005  0.8416212  1.2815516        Inf
```

Upon checking conditions, the simulated X's have little serial correlation and are roughly equivalent to the standard normal distribution, save a few slight deviations.

4) The Influence of the Correlation Coefficient ρ. Use your Gibbs sampling program from Exercise 3 to generate 15000 independent (x, y) realizations. Then, use these realizations to find the following:

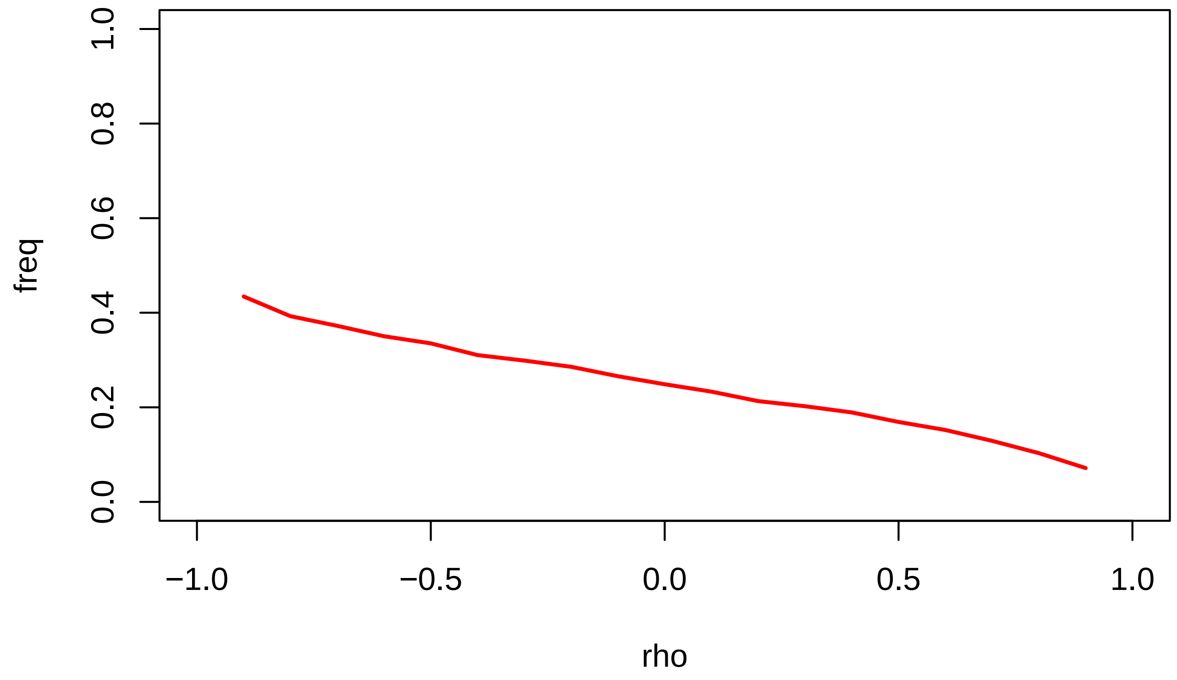
a) P($X > 0$ and $Y > 0$), for $\rho = \{-0.9, -0.8, \cdots, 0.8, 0.9\}$. Then plot these probabilities as a function of ρ. Set the vertical axis limits to (0, 1). R command: plot(rho.values, prob, xlim = c(-1,1), ylim = c(-1,1), ...).

```
rho <- seq(-.9, .9, .1)
freq <- rep(0, length(rho))

set.seed(540)
for (i in 1:length(rho)) {
  results4 <- BivariateBurnIn(15000, 0, 0, rho[i], lag = 5)
  counter <- 0
  for (j in 1:length(results4$X))
    if ((results4$X[j] > 0) & (results4$Y[j] > 0)) {
      counter <- counter + 1
    }
  freq[i] <- counter/length(results4$X)
}

plot(rho, freq, xlim = c(-1, 1), ylim = c(0, 1),
     type = 'l', col = 'red', lwd = 2)
```



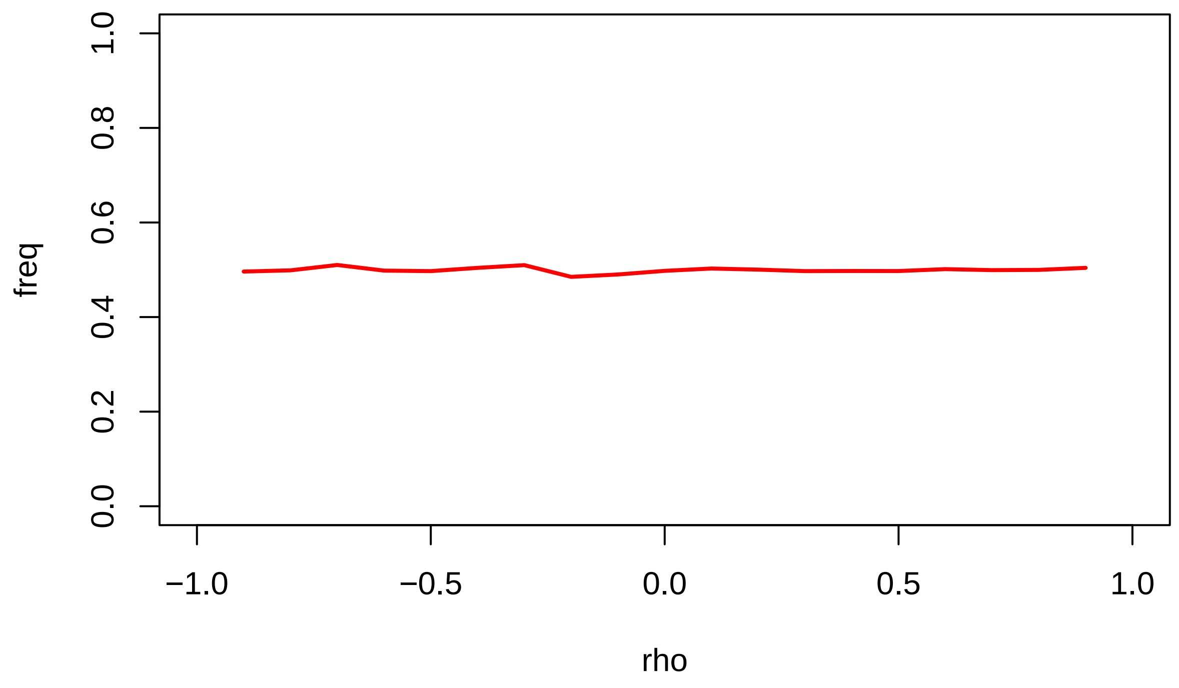
b) P($X < 0$ and $Y > 0$), for $\rho = \{-0.9, -0.8, \cdots, 0.8, 0.9\}$. Then plot these probabilities as a function of ρ. Set the vertical axis limits to (0, 1).

```
set.seed(545)
for (i in 1:length(rho)) {
  results4 <- BivariateBurnIn(15000, 0, 0, rho[i], lag = 5)
  counter <- 0
  for (j in 1:length(results4$X))
    if ((results4$X[j] < 0) & (results4$Y[j] > 0)) {
      counter <- counter + 1
    }
  freq[i] <- counter/length(results4$X)
}

plot(rho, freq, xlim = c(-1, 1), ylim = c(0, 1),
     type = 'l', col = 'red', lwd = 2)
```



c) $\mathrm{P}(X > 0)$, for $\rho = \{-0.9, -0.8, \cdots, 0.8, 0.9\}$. Then plot these probabilities as a function of ρ. Set the vertical axis limits to $(0, 1)$.

```
set.seed(550)
for (i in 1:length(rho)) {
  results4 <- BivariateBurnIn(15000, 0, 0, rho[i], lag = 5)
  counter <- 0
  for (j in 1:length(results4$X))
    if (results4$X[j] > 0) {
      counter <- counter + 1
    }
  freq[i] <- counter/length(results4$X)
}

plot(rho, freq, xlim = c(-1, 1), ylim = c(0, 1),
     type = 'l', col = 'red', lwd = 2)
```



5) The Slice Sampler – Suppose that we want to randomly sample from the set

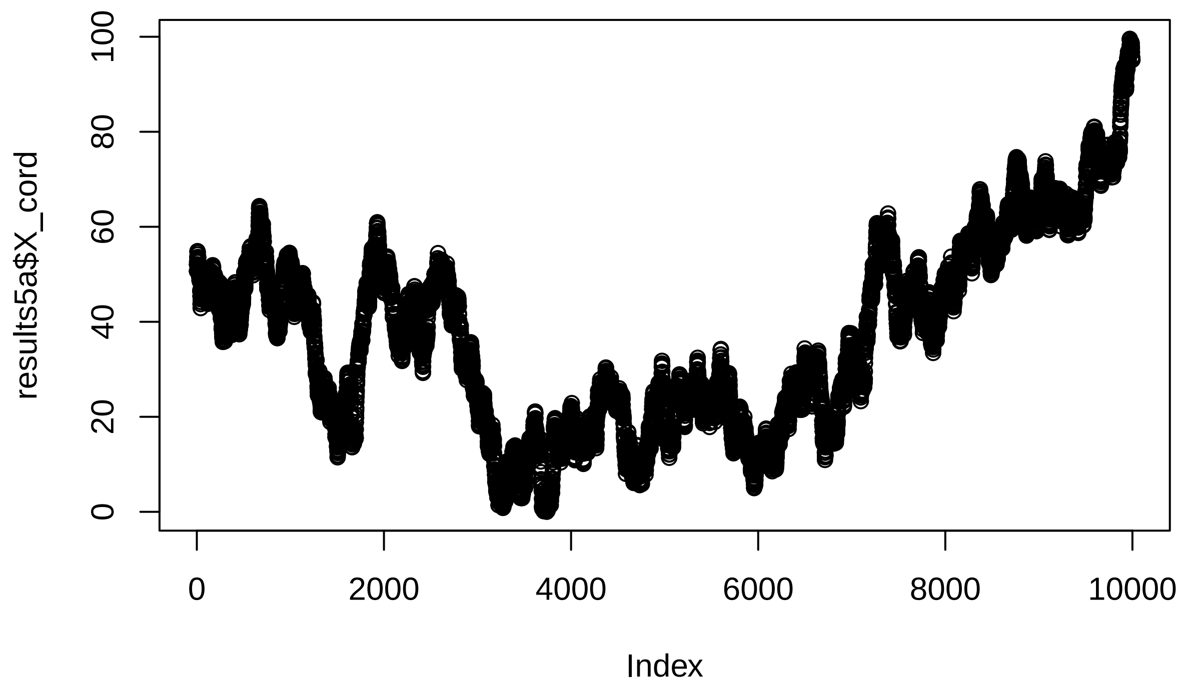
$$A = \{(x, y) \in \mathbb{R} | |x - y| \leq 1, 0 \leq x \leq 100, 0 \leq y \leq 100\}$$

a) Implement the slice sampler that we talked about in class. Start by choosing a random starting point and run the sampler for 100 iterations. Once it's working properly, run it for 10,000 iterations.

```r
SliceSampler <- function(iterations, initial_x, initial_y) {
  X <- c(); Y <- c()
  while (length(X) < iterations) {
    ## Initialize Values
    next_x <- initial_x; next_y <- initial_y

    ## Generate New Point
    next_x <- runif(1, next_y - 1, next_y + 1)
    next_y <- runif(1, next_x - 1, next_x + 1)

    ## If New Point is Valid, Append
    if ((abs(next_x - next_y) <= 1) & (next_x >= 0 & next_y >= 0) & (next_x <= 100 & next_y <= 100)) {
      X <- append(X, next_x); Y <- append(Y, next_y)
      initial_x <- next_x; initial_y <- next_y
    }
  }
  mylist <- list("X_cord" = X, "Y_cord" = Y)
  return(mylist)
  }

set.seed(500)
results5a <- SliceSampler(10000, 50, 50)
```

b) Make a plot of the values of X through time. This plot should give you an indication about the serial correlation that exists in a Gibbs Sampler. How does this plot compare to what you would expect of a 'random' set of points from this distribution? You don't have to actually implement another sampling algorithm to answer this question.

```r
plot(results5a$X_cord)
```



I would expect much more variation between points if the distribution was truly random. The x-coordinates are slow to change due to how they are calculated, meaning that over large spans of time, the x-coordinate changes little. i.e. This algorithm takes almost 4000 iterations for x to reach its minimum of 0, and the rest of the 6000 slowly increment x up to its maximum of 100. If it were random, the value of x would likely be much more uniformly distributed between these two extremes.

6) Consider the bivariate density

$$f(x,y) \propto \binom{n}{x} y^{x+a-1}(1-y)^{n-x+b-1}, x = 0, 1, \cdots, n, 0 \leq y \leq 1$$

It can be shown that for fixed a, b, and n, the conditional distributions are X|Y ~ Binomial(n,y) and Y|X ~ Beta(x+a, n-x+b). Use the Gibbs sampler to generate a chain with target density f(x,y) when a=2, b=4, and n=50. You can use either the built-in binomial and beta samplers or the ones that you made on an earlier assignment. Hint: i) Plot the values of X and Y (separately) through time after an appropriate burn-in and lag. What values did you choose for the burn-in and lag? Why? ii) Determine the mean of X and of Y.

```r
BetaBinom <- function(iterations, initial_x, initial_y, a = 2, b = 4, n = 50, burn_in = 5, lag = 9) {
  X <- c(); Y <- c(); counter <- 0
  next_x <- initial_x; next_y <- initial_y
  while (length(X) < iterations) {
    next_x <- rbinom(1, size = n, prob = next_y)
    next_y <- rbeta(1, next_x + a, n - next_x + b)
    if ((counter > burn_in) & (counter %% lag == 0)) {
      X <- append(X, next_x); Y <- append(Y, next_y)
      counter <- counter + 1
    }
    else {
      counter <- counter + 1
    }
  }
  mylist <- list("X_cord" = X, "Y_cord" = Y)
  return(mylist)
}

set.seed(405)
results6 <- BetaBinom(100, 50, (100/101))
```

After conducting an initial test to ensure the function was working correctly, I identified through a plot of values with the arrows command a burn-in of 5 should be used. Also, through the acf plot, a lag of 9 units should be used to avoid serial correlation. After doing so, the mean of X and Y are:

```r
mean(results6$X_cord); mean(results6$Y_cord)
```

```
## [1] 14.93
```

```
## [1] 0.3075386
```