Stat 102C HW5: Answer Key

Jonathan Arfa, Muzhou Liang

June 6, 2014

Problem 1.1

$$P(X_{t+1} = x, Y_t = y) = P(Y_t = y)P(X_{t+1} = x|Y_t = y) \tag{1}$$
$$= p_Y(y)p(x|y) \tag{2}$$
$$= p(x, y) \tag{3}$$

$$P(X_{t+1} = x, Y_{t+1} = y) = P(X_{t+1} = x)P(Y_{t+1} = y|X_{t+1} = x) \tag{4}$$
$$= p_X(x)p(y|x) \tag{5}$$
$$= p(x, y) \tag{6}$$

Thus, $(X_{t+1}, Y_t) \sim p(x, y)$ and $(X_{t+1}, Y_{t+1}) \sim p(x, y)$.

Problem 1.2

```
(probs = matrix(c(0.1, 0.4, 0.3, 0.2), nrow = 2, byrow = TRUE))

##      [,1] [,2]
## [1,]  0.1  0.4
## [2,]  0.3  0.2

# marginal probabilities of x: p(x==1), p(x==2)
(px = rowSums(probs))

## [1] 0.5 0.5

# marginal probabilities of y
(py = colSums(probs))

## [1] 0.4 0.6

# conditional probabilities of x/y==1: p(x==1/y==1), p(x==2/y==1)
(px_y1 = c(probs[1, 1], probs[2, 1])/sum(probs[, 1]))
```

```
## [1] 0.25 0.75

# conditional probabilities of x/y==2
(px_y2 = c(probs[1, 2], probs[2, 2])/sum(probs[, 2]))

## [1] 0.6667 0.3333

# conditional probabilities of y/x==1
(py_x1 = c(probs[1, 1], probs[1, 2])/sum(probs[1, ]))

## [1] 0.2 0.8

# conditional probabilities of y/x==2
(py_x2 = c(probs[2, 1], probs[2, 2])/sum(probs[2, ]))

## [1] 0.6 0.4
```

For the last part of this problem, we need to show that
$p(x, y) = P(X_{t+1} = x | Y_t = y)p_Y(y)$.
So, $p(x = 1, y = 1) = P(X_{t+1} = 1 | Y_t = 1)p_Y(1)$.

```
probs[1, 1]

## [1] 0.1

px_y1[1] * py[1]

## [1] 0.1
```

$p(x = 1, y = 2) = P(X_{t+1} = 1 | Y_t = 2)p_Y(2)$.

```
probs[1, 2]

## [1] 0.4

px_y2[1] * py[2]

## [1] 0.4
```

$p(x = 2, y = 1) = P(X_{t+1} = 2 | Y_t = 1)p_Y(1)$.

```
probs[2, 1]

## [1] 0.3

px_y1[2] * py[1]

## [1] 0.3
```

$$p(x = 2, y = 2) = P(X_{t+1} = 2|Y_t = 2)p_Y(2).$$

```
probs[2, 2]

## [1] 0.2

px_y2[2] * py[2]

## [1] 0.2
```

Problem 1.3

```
n = 5
# initialize a 3-dimensional array
jointdist = array(dim = c(2, 2, n + 1))
# set the first matrix of that array to p0
jointdist[, , 1] <- matrix(c(1, 0, 0, 0), nrow = 2)
for (t in 2:(n + 1)) {
    # get x_t|y_(t-1)
    x1y1 = px_y1[1] * sum(jointdist[, 1, t - 1])
    x2y1 = px_y1[2] * sum(jointdist[, 1, t - 1])
    x1y2 = px_y2[1] * sum(jointdist[, 2, t - 1])
    x2y2 = px_y2[2] * sum(jointdist[, 2, t - 1])
    jointdist[, , t] = matrix(c(x1y1, x1y2, x2y1, x2y2), 2, byrow = TRUE)

    # get y_t|x_t
    x1y1 = py_x1[1] * sum(jointdist[1, , t])
    x1y2 = py_x1[2] * sum(jointdist[1, , t])
    x2y1 = py_x2[1] * sum(jointdist[2, , t])
    x2y2 = py_x2[2] * sum(jointdist[2, , t])
    jointdist[, , t] = matrix(c(x1y1, x1y2, x2y1, x2y2), 2, byrow = TRUE)
}
for (t in 0:n) {
    print(paste("Joint distribution at t =", t))
    print(jointdist[, , t + 1])
}

## [1] "Joint distribution at t = 0"
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    0
## [1] "Joint distribution at t = 1"
##      [,1] [,2]
## [1,] 0.05  0.2
```

```
## [2,] 0.45  0.3
## [1] "Joint distribution at t = 2"
##         [,1]    [,2]
## [1,] 0.09167 0.3667
## [2,] 0.32500 0.2167
## [1] "Joint distribution at t = 3"
##         [,1]    [,2]
## [1,] 0.09861 0.3944
## [2,] 0.30417 0.2028
## [1] "Joint distribution at t = 4"
##         [,1]    [,2]
## [1,] 0.09977 0.3991
## [2,] 0.30069 0.2005
## [1] "Joint distribution at t = 5"
##         [,1]    [,2]
## [1,] 0.09996 0.3998
## [2,] 0.30012 0.2001
```

This isn't required by the homework, but we can create a metric of how close
our approximate joint distribution is to p at each value of t.

```r
distance = apply(jointdist, 3, function(mat) {
    # the distance between two matrices is the sqrt of the sum of the squared
    # differences
    sqrt(sum((mat - probs)^2))
})
plot(0:n, distance, xlab = "t", ylab = "", type = "o", pch = 19, main = "Distance from p")
```

Distance from p
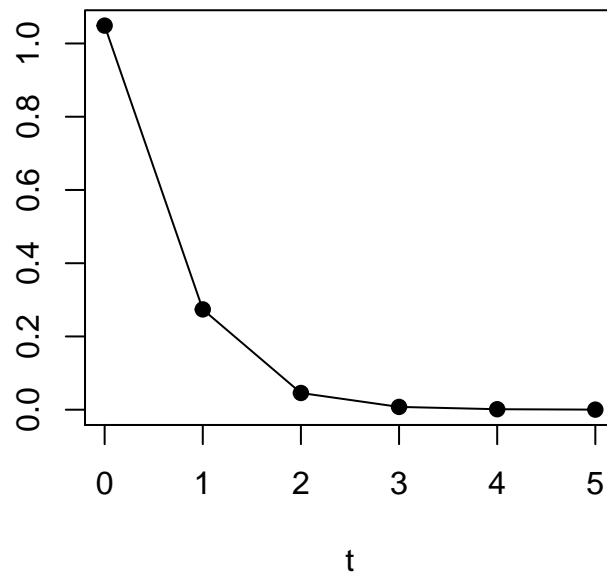
Problem 1.4

```r
Nsim = 20
M = 1e+05
# create a data frame of each person's x&y value
distribution = data.frame(x = rep(1, M), y = rep(1, M))
jointdist = array(0, dim = c(2, 2, Nsim))
for (t in 1:Nsim) {
    # get (x_t|y_t-1)
    r = runif(M)  #generate a random number for each person
    indy1 = which(distribution$y == 1)  #indices of which people have y_(t-1)==1
    indy2 = which(distribution$y == 2)
    distribution$x[indy1] = ifelse(r[indy1] < px_y1[1], 1, 2)
    distribution$x[indy2] = ifelse(r[indy2] < px_y2[1], 1, 2)

    # get (y_t|x_t)
    r = runif(M)
    indx1 = which(distribution$x == 1)
    indx2 = which(distribution$x == 2)
```

```
    distribution$y[indx1] = ifelse(r[indx1] < py_x1[1], 1, 2)
    distribution$y[indx2] = ifelse(r[indx2] < py_x2[1], 1, 2)
    for (x in 1:2) for (y in 1:2) {
        # fill in each cell of the joint dist.
        jointdist[x, y, t] = sum(distribution$x == x & distribution$y == y)
    }
    # convert from number of people to proportion
    jointdist[, , t] = jointdist[, , t]/M
}
jointdist[, , Nsim]

##         [,1]    [,2]
## [1,] 0.09889 0.3968
## [2,] 0.30230 0.2021

probs

##      [,1] [,2]
## [1,]  0.1  0.4
## [2,]  0.3  0.2

distance = apply(jointdist, 3, function(mat) {
    sqrt(sum((mat - probs)^2))
})
plot(1:Nsim, distance, xlab = "t", ylab = "", type = "o", pch = 19, main = paste0("Distance
    M))
axis(1, at = c(1, seq(5, Nsim, by = 5)))
```
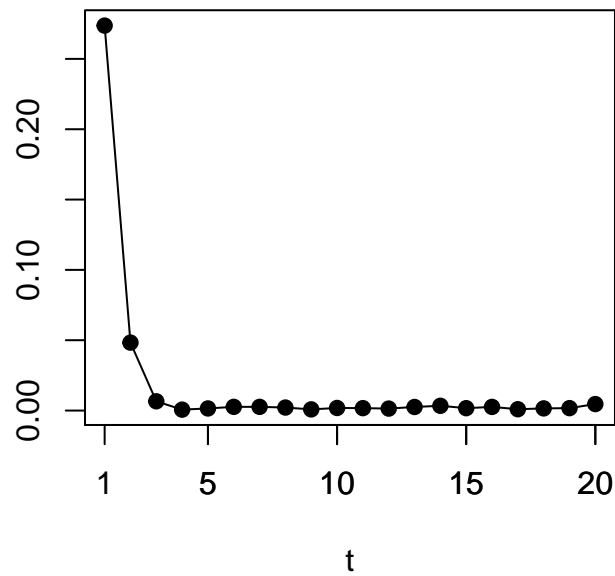
Distance from p with M = 1e+05

Problem 2

Since we want to test out various values of ρ, let's create a function that we can
run with different parameters.

```
gibbsBivarNorm = function(rho, Nsim = 2000, burnin = 1000, x0 = 0, y0 = 0, returnvals = TRUE
    plotit = TRUE) {
    x = y = numeric(Nsim)
    # first iteration happens outside the for loop
    x[1] = rnorm(1, rho * y0, sqrt(1 - rho^2))
    y[1] = rnorm(1, rho * x[1], sqrt(1 - rho^2))
    for (t in 2:Nsim) {
        x[t] = rnorm(1, rho * y[t - 1], sqrt(1 - rho^2))  #conditional dist. of x/y
        y[t] = rnorm(1, rho * x[t], sqrt(1 - rho^2))  #conditional dist. of y/x
    }
    # remove the samples from the burn-in period.
    x = x[(burnin + 1):Nsim]
    y = y[(burnin + 1):Nsim]
    if (plotit)
        plot(x, y, main = paste("rho =", rho), xlim = c(-4, 4), ylim = c(-4,
```

```
            4))
    # note that I'm keeping the limits of the graph constant regardless of rho
    if (returnvals)
        return(list(x = x, y = y, expec_val = mean(sqrt(x^2 + y^2))))
}
par(mfrow = c(2, 2), mar = c(2.5, 2, 2, 1) + 0.1)
# FYI, default margins are mar=c(5, 4, 4, 2) + 0.1
gibbsBivarNorm(-0.9)$expec_val

## [1] 1.177

gibbsBivarNorm(-0.3)$expec_val

## [1] 1.254

gibbsBivarNorm(0.1)$expec_val

## [1] 1.276

gibbsBivarNorm(0.9)$expec_val

## [1] 1.16
```
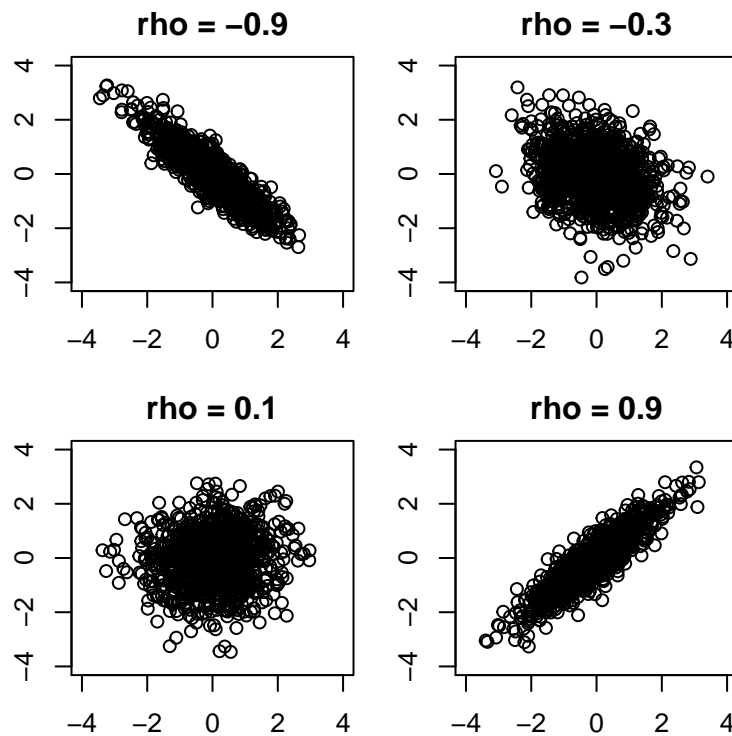
Problem 2-Bonus

We'll create a function that outputs a series of plots, interrupted by pauses. If we wrap this function in the saveGIF() function it should create an animated gif.

The following code chunk isn't run in this file, but try it in your R console.

```r
library(animation)
gibbsBivarNorm_walk = function(rho, M, x0 = 0, y0 = 0, Nsim = 50, burnin = 0,
    arrow = TRUE) {
    xvals = yvals = matrix(nrow = Nsim + 1, ncol = M)
    for (i in 1:M) {
        # get the samples from each walk separately
        samples = gibbsBivarNorm(rho, Nsim, burnin, x0, y0, plotit = FALSE)
        xvals[, i] = c(x0, samples$x)
        yvals[, i] = c(y0, samples$y)
    }
    for (t in (burnin + 1):(Nsim + 1)) {
        # plot all samples from a given t together
        plot(xvals[t, ], yvals[t, ], type = "p", col = 1:M, xlim = c(-4, 4),
            ylim = c(-4, 4), xlab = "x", ylab = "y", pch = 16, main = paste("rho =",
                rho))
        # draw arrows from previous points
        if (arrow & t > 1)
            arrows(xvals[t - 1, ], yvals[t - 1, ], xvals[t, ], yvals[t, ], length = 0.1,
                col = "grey")
        # write t (notice that I'm placing the text where it's less likely to
        # overlap text)
        if (rho >= 0) {
            text(-3, 3, paste("t =", t - 1), cex = 1.5)
        } else text(3, 3, paste("t =", t - 1), cex = 1.5)
        ani.pause()
    }
}

par(mfrow = c(1, 1), mar = c(5, 4, 2, 2) + 0.1)
ani.options(interval = 0.1)
# the next 2 lines will generate the 'video' in your plotting window small
# M. See how the individual points are moving around?
gibbsBivarNorm_walk(rho = 0.9, M = 5)
# big M. See how the distribution is coverging to the joint dist. of (x,y)?
gibbsBivarNorm_walk(rho = 0.9, M = 500, arrow = FALSE, Nsim = 25)
gibbsBivarNorm_walk(rho = 0.95, M = 500, arrow = FALSE, x0 = -3, y0 = -3, Nsim = 25)
# And the following line creates the .gif.  The file should pop up
# separately, make sure you save it somewhere useful.
saveGIF(gibbsBivarNorm_walk(rho = 0.9, M = 5), movie.name = "smallM.gif")
```

```r
saveGIF(gibbsBivarNorm_walk(rho = 0.9, M = 500, arrow = FALSE), movie.name = "bigM.gif")
saveGIF(gibbsBivarNorm_walk(rho = 0.95, M = 500, arrow = FALSE, x0 = -3, y0 = -3),
    movie.name = "bigM2.gif")
```