Homework 9

Steve Harms

November 3, 2017

Exercise 1

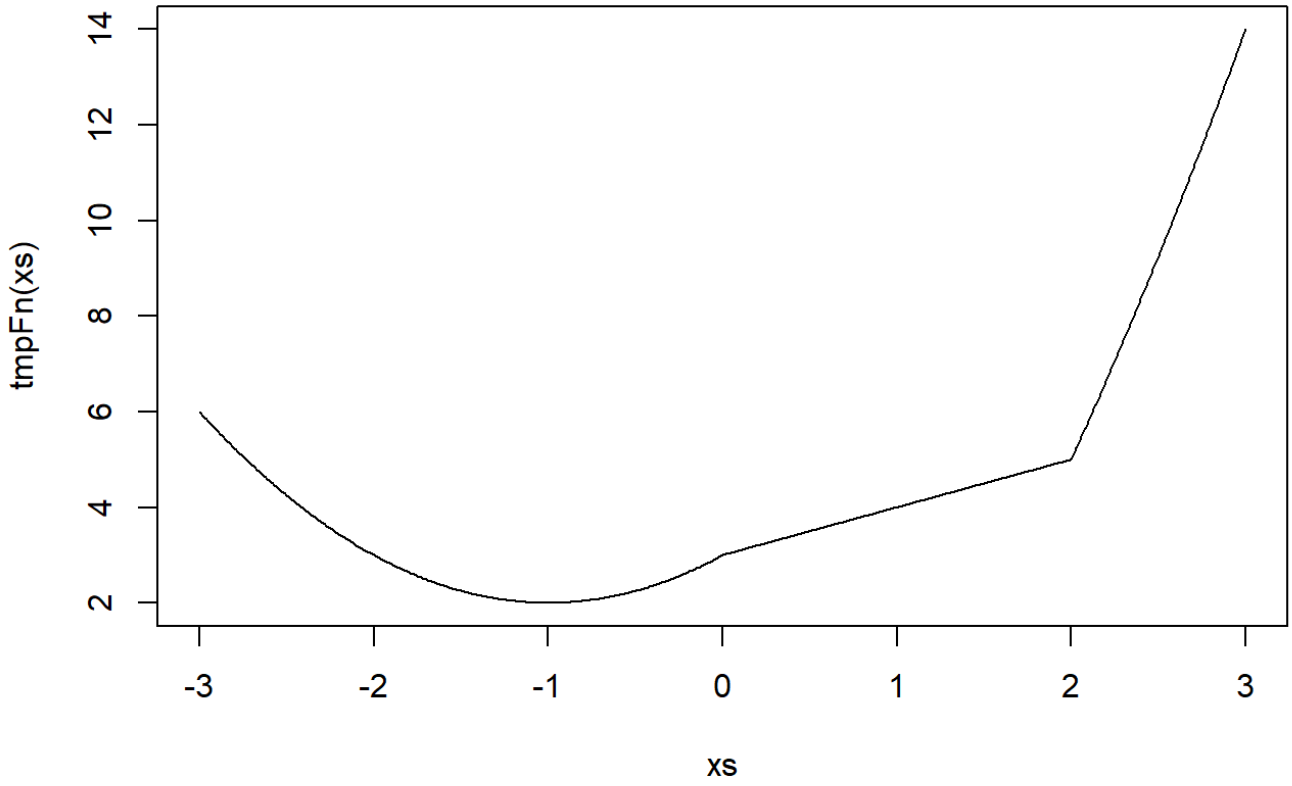
```
#the function makes a diagonal matrix with k
#then changes diagonal-adjacent entries to 1
blockdiag <- function(n,k){
  mat <- k*diag(n)
  for (i in 1:n){
    mat[i,i-1] = 1
    mat[i-1, i] = 1
  }
  return(mat)
}
#test the function with n = 6, k = 5
blockdiag(6,5)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    5    1    0    0    0    0
## [2,]    1    5    1    0    0    0
## [3,]    0    1    5    1    0    0
## [4,]    0    0    1    5    1    0
## [5,]    0    0    0    1    5    1
## [6,]    0    0    0    0    1    5
```

Exercise 2

```
#The function just evaluates each element based on its value
#then returns a vector of f(x)
tmpFn <- function(xVec) {
  fx <- NULL
  for (i in 1:length(xVec)){
    x <- xVec[i]
    if (x < 0) fx[i] = x^2 + 2*x + 3 else
      if (x < 2) fx[i] = x + 3 else
        fx[i] = x^2 + 4*x - 7
  }
  return(fx)
}

#Generate some values
xs <- seq(from = -3, to = 3, by = .01)
#Then plot those values
plot(x = xs, y = tmpFn(xs), type = 'l')
```



Exercise 3

```
#a function to find gcd
#we know it's the gcd when the remainder is 0
gcd <- function(m,n) {
  remainder <- m%%n
  gcd <- n
  n <- remainder
  while (remainder != 0) {
    remainder <- gcd %% remainder
    gcd <- n
    n <- remainder
  }
  return(gcd)
}

#Test on some values
gcd(78564,35148)
```

```
## [1] 12
```

```
gcd(35,28)
```

```
## [1] 7
```

Exercise 4

```
#The function sorts the entries of the matrix
#Then one by one adds the array indices into the output matrix
order.matrix <- function(x){
  outmat <- which(x == max(x), arr.ind = T)
  sx <- sort(x, decreasing = T)
  for (i in 2:length(x)){
    outmat <- rbind(outmat,which(x == sx[i], arr.ind = T))
  }
  return(outmat)
}
#Make a matrix to test the function
testchisq <- matrix(rchisq(12, df = 1), nrow = 4, ncol = 3)
testchisq
```

```
##             [,1]      [,2]      [,3]
## [1,] 1.10109377 0.4698605 0.2274961
## [2,] 0.04590663 3.8976063 0.1012267
## [3,] 0.08194791 1.0299384 0.4280722
## [4,] 0.02495989 0.8888305 0.1677501
```

```
#Our result from calling the function
order.matrix(testchisq)
```

```
##       row col
##  [1,]   2   2
##  [2,]   1   1
##  [3,]   3   2
##  [4,]   4   2
##  [5,]   1   2
##  [6,]   3   3
##  [7,]   1   3
##  [8,]   4   3
##  [9,]   2   3
## [10,]   3   1
## [11,]   2   1
## [12,]   4   1
```

Exercise 5

a)

For the polar coordinate conversion, I used the algorithm from https://en.wikipedia.org/wiki/N-sphere#Spherical_coordinates, which only works if at least one element of x is non-zero. It can also be done with repeated products as in Prof. Maitra's problem statement.

```
#A function
polaroid <- function(x){
#first get the R
  R <- sqrt(t(x)%*%x)
#then theta 1
  theta <- c(acos(x[1]/R))
  p <- length(x)
#a loop to calculate theta2 to theta(n-2)
  for (i in 2:(p-2)){
    divis <- sqrt(t(x[i:p])%*%x[i:p])
    theta[i] <- acos(x[i]/divis)
  }
#the final theta depends on whether xn is >= or < 0
  if(x[p] >= 0) thn1 <- acos(x[p-1]/(sqrt(x[p]^2+x[p-1]^2))) else
    thn1 <- 2*pi - acos(x[p-1]/(sqrt(x[p]^2+x[p-1]^2)))
  theta[p-1] <- thn1
#return the thetas and R together
  outvect <- c(R, theta)
  return(t(outvect))
}

#a quick test of the function on a simple 3-d cartesian coordinate
testp <- c(1,sqrt(6)/2, sqrt(6)/2)
polaroid(testp)
```

```
##      [,1]     [,2]      [,3]
## [1,]    2 1.047198 0.7853982
```

b)

```
#A quick function to calculate sum of squares
#then divide each element by the sqrt to get normalized
normalize <- function(x){
  SoS <- sqrt(sum(x^2))
  normed <- x / SoS
  return(normed)
}
#a quick test of the function
testn <- c(1,2,3,4,5)
t(normalize(testn))%*%normalize(testn)
```

```
##      [,1]
## [1,]    1
```

c)

```r
#generate the matrix of psuedo random numbers
pseud <- matrix(rnorm(5000), ncol = 5, nrow = 1000)
#apply normalize to the rows
z <- t(apply(pseud, MARGIN = 1, FUN = normalize))
#some checks
head(pseud)
```

```
##             [,1]        [,2]        [,3]        [,4]        [,5]
## [1,] -0.94526249  0.59671200  1.52404237 -0.33801130  0.04568545
## [2,] -1.58525875  0.64756178  0.34364386  1.38974848  0.58279551
## [3,]  0.54684333  0.05856602  0.34123390 -0.51970283 -0.35151340
## [4,]  1.77511634  2.13201903  0.41747490 -2.03975000  1.25093198
## [5,] -0.24586427 -0.10637914 -0.09252424  0.48892893  0.05170181
## [6,]  0.03081857 -0.17768356  0.41757960  0.06753508 -1.92629222
```

```r
head(z)
```

```
##             [,1]        [,2]       [,3]        [,4]        [,5]
## [1,] -0.49217535  0.31069353  0.7935321 -0.17599432  0.02378731
## [2,] -0.68719840  0.28071343  0.1489672  0.60244609  0.25263771
## [3,]  0.60664548  0.06497073  0.3785508 -0.57653692 -0.38995449
## [4,]  0.48141256  0.57820478  0.1132194 -0.55318136  0.33925347
## [5,] -0.43324450 -0.18745373 -0.1630396  0.86155570  0.09110525
## [6,]  0.01556165 -0.08972022  0.2108542  0.03410142 -0.97266937
```

```r
#use apply with the K-S test across columns of z
#In all 5 cases, we reject the null hypothesis
apply(z, MARGIN = 2, FUN = ks.test, "punif", min=-1, max = 1)
```

```
## [[1]]
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  newX[, i]
## D = 0.098333, p-value = 7.986e-09
## alternative hypothesis: two-sided
##
##
## [[2]]
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  newX[, i]
## D = 0.10113, p-value = 2.618e-09
## alternative hypothesis: two-sided
##
##
## [[3]]
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  newX[, i]
## D = 0.12259, p-value = 1.772e-13
## alternative hypothesis: two-sided
##
##
## [[4]]
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  newX[, i]
## D = 0.10397, p-value = 8.175e-10
## alternative hypothesis: two-sided
##
##
## [[5]]
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  newX[, i]
## D = 0.11095, p-value = 4.07e-11
## alternative hypothesis: two-sided
```

In all 5 cases, we reject the null hypothesis that the columns are uniformly distributed. It should be obvious that this is the case since we generated normal r.v.s, not uniforms.

d)

```
#apply polaroid across rows of z
y <- t(apply(pseud, MARGIN = 1, FUN = polaroid))
head(y)
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]
## [1,] 1.9205807 2.0853833 1.2058325 0.2201745 3.0072472
## [2,] 2.3068429 2.3284219 1.1740629 1.3465982 0.3970781
## [3,] 0.9014216 0.9189622 1.4889781 1.0726712 3.7362855
## [4,] 3.6873079 1.0685307 0.8504047 1.3980625 2.5914677
## [5,] 0.5674954 2.0188859 1.7803128 1.7568102 0.1053535
## [6,] 1.9804183 1.5552341 1.6606483 1.3574476 4.7474343
```

```
#K-S test for R^2, which is the first element of each row squared
ks.test(y[,1]^2, "pchisq", df = 5)
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  y[, 1]^2
## D = 0.03125, p-value = 0.2829
## alternative hypothesis: two-sided
```

```
#K-S test for the thetas
ks.test(y[,2], "punif", min = 0, max = pi)
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  y[, 2]
## D = 0.19409, p-value < 2.2e-16
## alternative hypothesis: two-sided
```

```
ks.test(y[,3], "punif", min = 0, max = pi)
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  y[, 3]
## D = 0.16377, p-value < 2.2e-16
## alternative hypothesis: two-sided
```

```
ks.test(y[,4], "punif", min = 0, max = pi)
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  y[, 4]
## D = 0.12359, p-value = 1.084e-13
## alternative hypothesis: two-sided
```

```
ks.test(y[,5], "punif", min = 0, max = 2*pi)
```

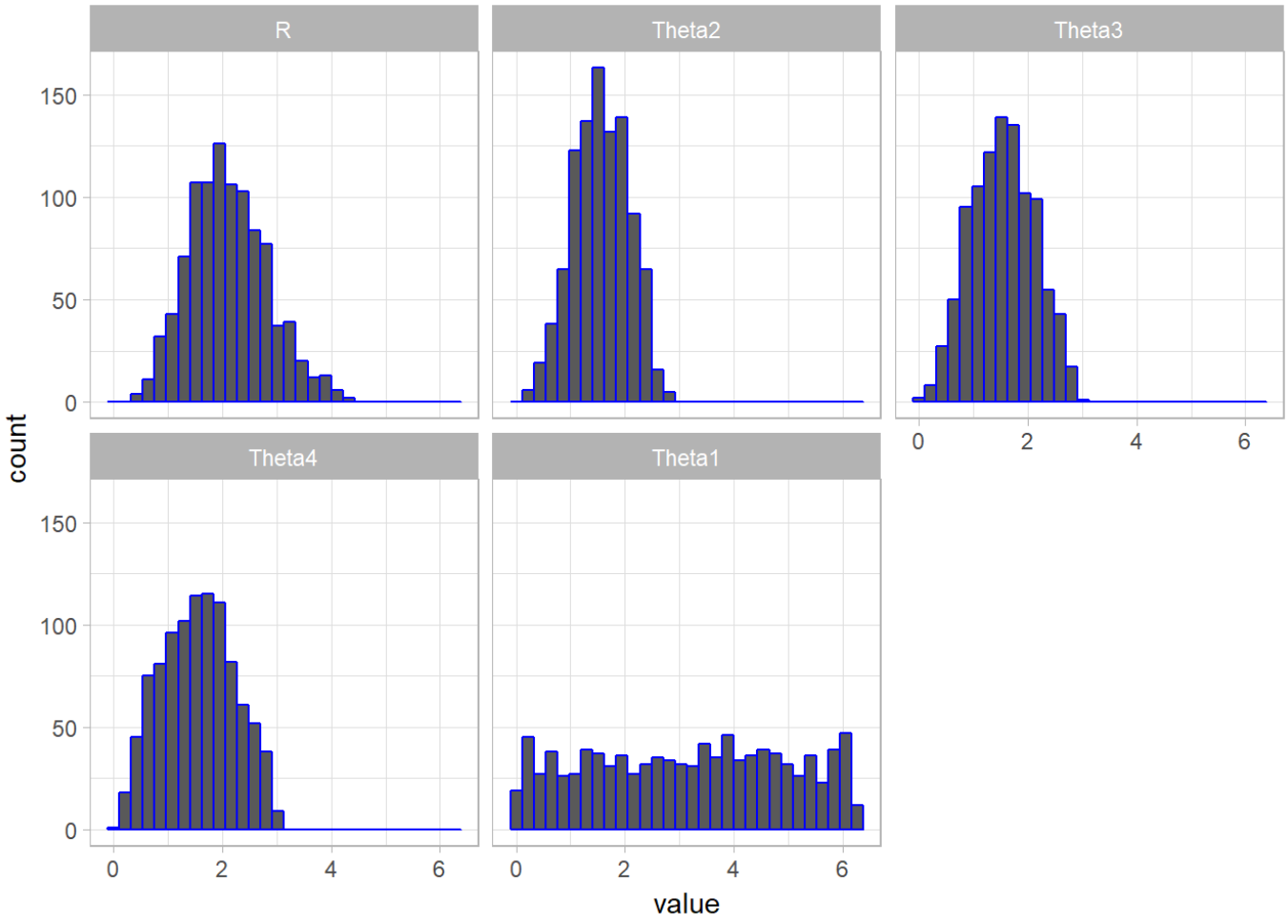
```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  y[, 5]
## D = 0.019231, p-value = 0.8533
## alternative hypothesis: two-sided
```

```
#A plot for each theta, columns 2-5
y <- as.data.frame(y)
names(y) <- c("R", "Theta2", "Theta3", "Theta4", "Theta1")
melty <- melt(y)
```

```
## No id variables; using all as measure variables
```

```
g <- ggplot(data = melty, aes(x = value))
h <- g + geom_histogram(color = "blue") + facet_wrap(~variable, nrow = 2) + theme_
light()
h
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



For all 3 thetas, we reject the null hypothesis of uniform distribution, using the same reasoning as in part(c), and it's also visible in the plots that they are normal not uniform. However, we can see that R^2 does in fact belong to chi-squared distribution, which makes since because it is computed by adding squared normal random variables. Theta 1 also is Uniform on (0, 2pi), which is apparent from

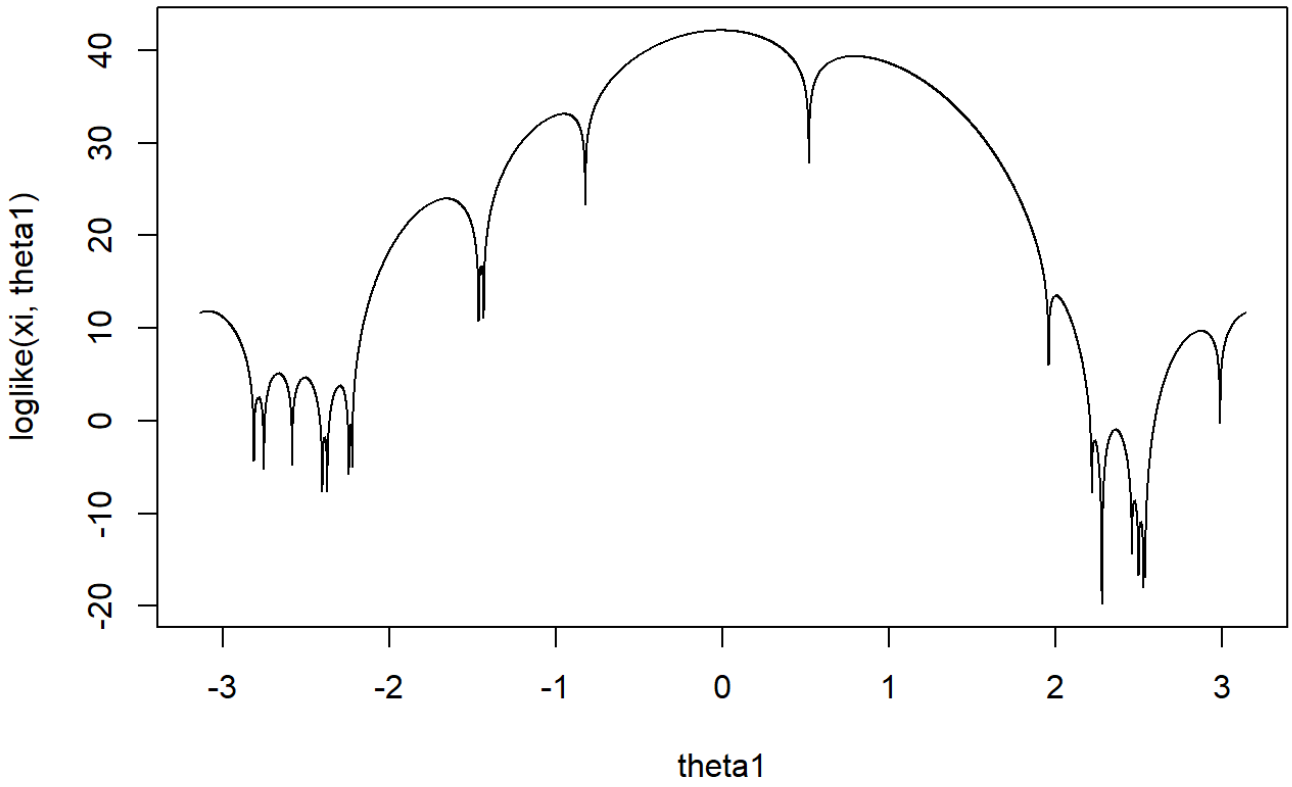
the histogram, and we know this should be true based on how it's calculated, which is just a normal rv divided by a sum of squares of normal rvs.

Exercise 6

a)

```
#first, a function to calculate log-likelihood
loglike <- function(x,theta){
  n <- length(x)
  lth <- n*log(2*pi)
  i = 1
  while (i <= n) {
    lth <- lth + log(1 - cos(x[i] - theta))
    i <- i + 1
  }
  return(lth)
}

#now generate our observed random sampmle
xi <- c(3.91, 4.85, 2.28, 4.06, 3.70, 4.04, 5.46, 3.53, 2.28,
        1.96, 2.53, 3.88, 2.22, 3.47, 4.82, 2.46, 2.99, 2.54, .52, 2.5)
theta1 <- seq(from = -1*pi, to = pi, by = .001)
#plot them
plot(x = theta1, y = loglike(xi, theta1), type = 'l')
```



b)

```
#Now use optimize() to find the global maximum of theta
#It's near x = 0, f(x) = 40, as expected based on the plot
optimize(f = loglike, interval = c(-pi, pi), x = xi, maximum = TRUE)
```

```
## $maximum
## [1] -0.0119724
##
## $objective
## [1] 42.17217
```

c)

```r
#the newton() function from class
newton <- function(fun, derf, x0, eps){
  iter <- 0
  repeat {
    iter <- iter + 1
    x1 <- x0 - fun(x0) / derf(x0)
    if (abs(x0 - x1) < eps || abs(fun(x1)) < 1e-10)
    break
    x0 <- x1
    cat("****** Iter. No: ", iter, "Current Iterate = ", x1, fill=T)
  }
return(x1)
}

#Now, a bunch of functions of first and second derivatives
dll <- function(x, theta){
  n <- length(x)
  d.l <- 0
  i <- 1
  while(i<=n){
    d.l <- d.l-sin(x[i] - theta)/ (1-cos(x[i] - theta))
    i<- i + 1
  }
  return(d.l)
}

d2ll <- function(x, theta){
  n <- length(x)
  d2l <- 0
  i <- 1
  while(i<=n){
    d2l <- d2l- 1/(1-cos(x[i] - theta))
    i<- i + 1
  }
  return(d2l)
}

dllth <- function(theta){
  dll(xi, theta)
}
d2llth <- function(theta){
  d2ll(xi, theta)
}

#finally, our result, which is similar to the result in (b) since we started nearby
newton(dllth, d2llth, x0 = 0, eps = .00001)
```

```
## ****** Iter. No:  1 Current Iterate =  -0.01191193
## ****** Iter. No:  2 Current Iterate =  -0.011972
```

```
## [1] -0.011972
```

d)

```
#call the function using different starting points
newton(dllth, d2llth, x0 = -2, eps = .00001)
```

```
## ****** Iter. No:  1 Current Iterate =  -1.756154
## ****** Iter. No:  2 Current Iterate =  -1.641367
## ****** Iter. No:  3 Current Iterate =  -1.657301
## ****** Iter. No:  4 Current Iterate =  -1.65828
```

```
## [1] -1.658283
```

```
newton(dllth, d2llth, x0 = -2.7, eps = .00001)
```

```
## ****** Iter. No:  1 Current Iterate =  -2.674114
## ****** Iter. No:  2 Current Iterate =  -2.666794
## ****** Iter. No:  3 Current Iterate =  -2.6667
```

```
## [1] -2.6667
```

We see different results because the Newton-Raphson method only finds the local maximums, not global. We can see from the plot that this function has several local maximums, so the function will find whichever local maximum it is closest to.

Exercise 7

a)

```
#Generate the men and women
men = rnorm(n= 100, 125, 25)
fm = rnorm(n= 100, 125, 15)
#combine them into a data frame
firstgen <- data.frame(men,fm)
#add columns for average height and generation to make it easier later
firstgen$ht <- apply(firstgen, MARGIN = 1, FUN = mean)
firstgen$gen <- rep(1, times = length(firstgen$men))
head(firstgen)
```

```
##        men       fm       ht gen
## 1 174.2248 122.5122 148.3685   1
## 2 102.8627 159.2612 131.0619   1
## 3 151.4831 106.6116 129.0474   1
## 4 147.1024 141.3545 144.2285   1
## 5 154.5626 133.7558 144.1592   1
## 6 141.8742 114.6681 128.2711   1
```

b)

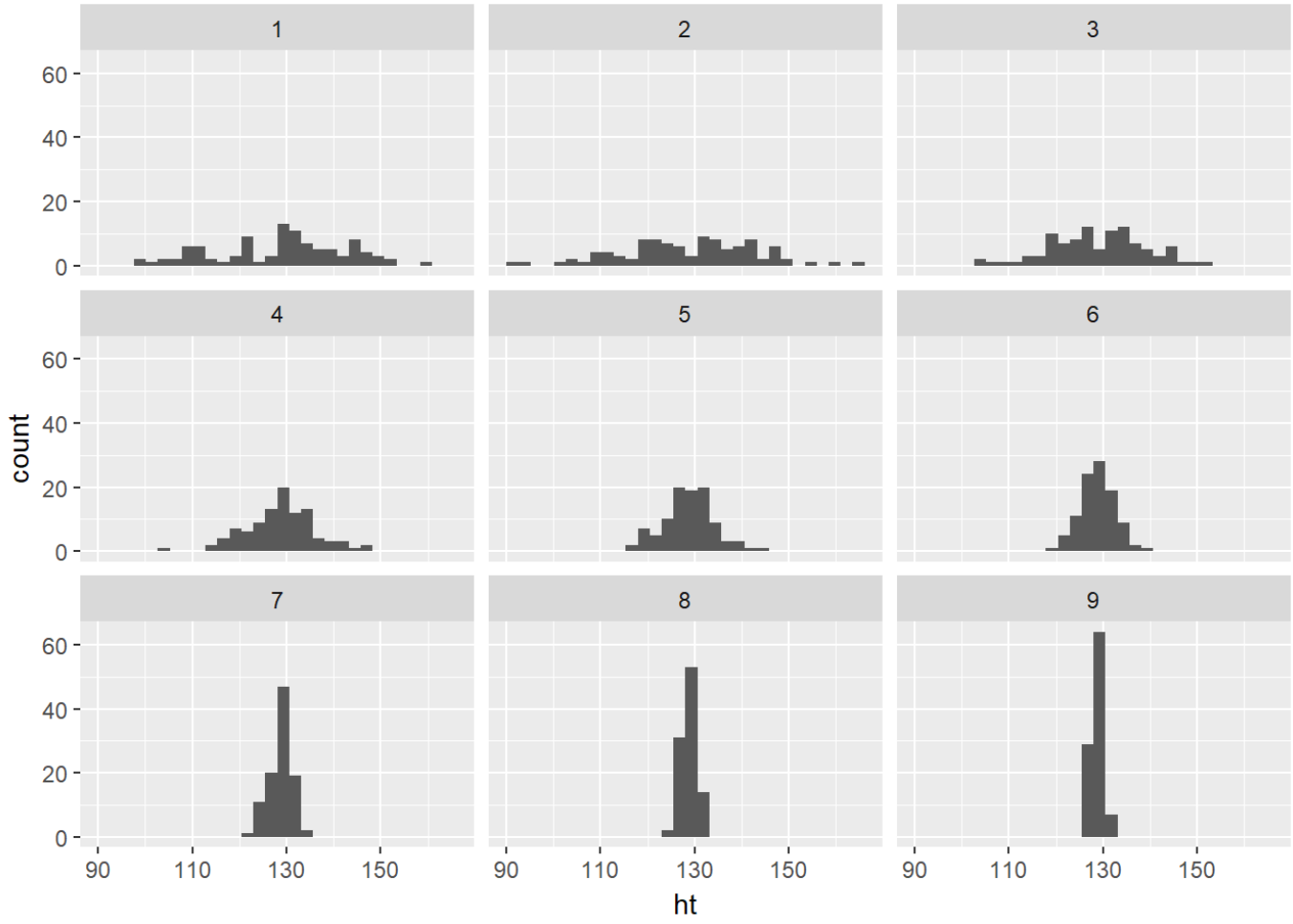
```r
#a function to generate the next generation
nextgen <- function(m,fem, gen){
  men <- c(sample(m, size = length(m), replace = FALSE))
  fems<- fem
  #make pairs
  pair <- cbind(men, fems)
  #next generation is the average of the pairs
  height <- apply(pair, MARGIN = 1, FUN = mean)
  #throw them into a data frame and output with generation #
  nextgen <- data.frame(cbind(height, height, height))
  names(nextgen) <- c("men", "fm", "ht")
  nextgen$gen <- rep(gen, times = length(m))
  return(nextgen)
}
```

c)

```r
#generate 9 generations starting with our first
gen2 <- nextgen(firstgen$men, firstgen$fm, 2)
gen3 <- nextgen(gen2$men, gen2$fm, 3)
gen4 <- nextgen(gen3$men, gen3$fm, 4)
gen5 <- nextgen(gen4$men, gen4$fm, 5)
gen6 <- nextgen(gen5$men, gen5$fm, 6)
gen7 <- nextgen(gen6$men, gen6$fm, 7)
gen8 <- nextgen(gen7$men, gen7$fm, 8)
gen9 <- nextgen(gen8$men, gen8$fm, 9)

#put all of the generations into a data frame to make it easy to plot
#the histograms show average height clearly revert closer to the mean with each gen
eration
allgens <- rbind.data.frame(firstgen, gen2, gen3, gen4, gen5, gen6, gen7, gen8, gen
9)
g <- ggplot(data = allgens, aes(x=ht))
h <- g + geom_histogram() + facet_wrap(~factor(gen), nrow = 3)
h
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Exercise 8

a)

```
#a function to read in the cluster data files
readclust <- function(x, sort.obs = FALSE){
  #read in the file
  readin <- readLines(x)
  #find the total # of clusters
  nclust <- as.numeric(substring(readin[1], first = regexpr("\\d", readin[1])))
  #find the location of the cluster headings in the file
  clust.titles <- which(regexpr("size", readin)>0)
  clustnum <- c(0)
  clustsize <- c(0)
  obs <- c(rep(NA, times = 2))
  grp <- c(rep(NA, times = 2))

  #a loop to find all of our clusters and their sizes in the file
  for (i in 1:(length(clust.titles))){
    tind <- clust.titles[i]
    clustnum[i] <- as.numeric(substring(readin[tind], first = 9, last = regexpr(";
", readin[tind])-2))
    clustsize[i] <- as.numeric(substring(readin[tind], first = regexpr("size=", re
adin[tind])+5))
  }

  #a loop to assign a cluster number to each observation, and convert the observati
on to integer format
  for (i in 1:(length(clustnum))){
    clind <- clust.titles[i] + 1
    for (j in clind:(clind + clustsize[i])){
      grp[j-2*i-1] <- clustnum[i]
      obs[j-2*i-1] <- as.integer(readin[j])
    }
  }
  clusters <- data.frame(observation = obs, grp)
  #filter out NAs
  #also sort the observations
  cluster <- clusters %>%filter(observation >= 0) %>% filter(grp >= 0)
  if (sort.obs == T) cluster <- arrange(cluster, observation)
  return(cluster)
}
```

b)

```
#function calls for each of the files
iris1 <- readclust("Iris1.out")
head(iris1)
```

```
##   observation grp
## 1         100   0
## 2         102   0
## 3         103   0
## 4         104   0
## 5         105   0
## 6         107   0
```

```
iris2 <- readclust("Iris2.out")
head(iris2)
```

```
##   observation grp
## 1          50   0
## 2          51   0
## 3          52   0
## 4          53   0
## 5          54   0
## 6          55   0
```

```
#can also sort them
iris1.sort <- readclust("Iris1.out", sort.obs = T)
head(iris1.sort)
```

```
##   observation grp
## 1           0   1
## 2           1   1
## 3           2   1
## 4           3   1
## 5           4   1
## 6           5   1
```

```
iris2.sort <- readclust("Iris2.out", sort.obs = T)
head(iris2.sort)
```

```
##   observation grp
## 1           0   1
## 2           1   1
## 3           2   1
## 4           3   1
## 5           4   1
## 6           5   1
```

```
#a quick check of our sorted data
head(cbind(iris1.sort, iris2.sort$grp))
```

```
##   observation grp iris2.sort$grp
## 1           0   1               1
## 2           1   1               1
## 3           2   1               1
## 4           3   1               1
## 5           4   1               1
## 6           5   1               1
```

```
tail(cbind(iris1.sort, iris2.sort$grp))
```

```
##     observation grp iris2.sort$grp
## 145         144   0               0
## 146         145   0               0
## 147         146   2               0
## 148         147   0               0
## 149         148   0               0
## 150         149   2               0
```

```r
#cross tabulation
#matches what's in the file
table("iris1" = iris1.sort$grp, "iris2" = iris2.sort$grp)
```

```
##       iris2
## iris1  0  1
##     0 37  0
##     1  0 50
##     2 63  0
```