HW 6

Steve Harms

October 15, 2017

1)

a)

```
h.for <- function(x, n){
  h = 0
  for (i in 0:n){
    h = h + x^i
  }
  return(h)
}
```

b)

```
#next, try it with a while loop
h.while <- function(x, n){
  h = 0
  i = 0
  while (i < n + 1){
    h = h + x^i
    i = i+1
  }
  return(h)
}
```

c)

```
#calculate computing time for each of the combinations of x and n, for each functio
n (8 total)
x.3af <- system.time(h.for(.3, 500))
x.3bf <- system.time(h.for(.3, 5000))
x.101af <- system.time(h.for(1.01, 500))
x.101bf <- system.time(h.for(1.01, 5000))
x.3aw <- system.time(h.while(.3, 500))
x.3bw <- system.time(h.while(.3, 5000))
x.101aw <- system.time(h.while(1.01, 500))
x.101bw <- system.time(h.while(1.01, 5000))

#report the "user" time to compute, which is the first element in the object return
ed by system.time()
loop.time <- rbind(x.3af, x.3bf, x.101af, x.101bf, x.3aw, x.3bw, x.101aw, x.101bw)
[,1]
loop.time
```

```
##   x.3af   x.3bf x.101af x.101bf   x.3aw   x.3bw x.101aw x.101bw
##    0.00    0.01    0.00    0.00    0.00    0.00    0.00    0.00
```

d)

```
h.no <- function(x,n){
  ex <- c(0:n)
  mat <- rep(x, times = n + 1)
  sum(mat^ex)
}
x.3afn <- system.time(h.no(.3, 500))
x.3bfn <- system.time(h.no(.3, 5000))
x.101afn <- system.time(h.no(1.01, 500))
x.101bfn <- system.time(h.no(1.01, 5000))

noloop.time <-rbind(x.3afn, x.3bfn, x.101afn, x.101bfn)[,1]
#Compare computing times
loop.time
```

```
##   x.3af   x.3bf x.101af x.101bf   x.3aw   x.3bw x.101aw x.101bw
##    0.00    0.01    0.00    0.00    0.00    0.00    0.00    0.00
```

```
noloop.time
```

```
##   x.3afn   x.3bfn x.101afn x.101bfn
##        0        0        0        0
```

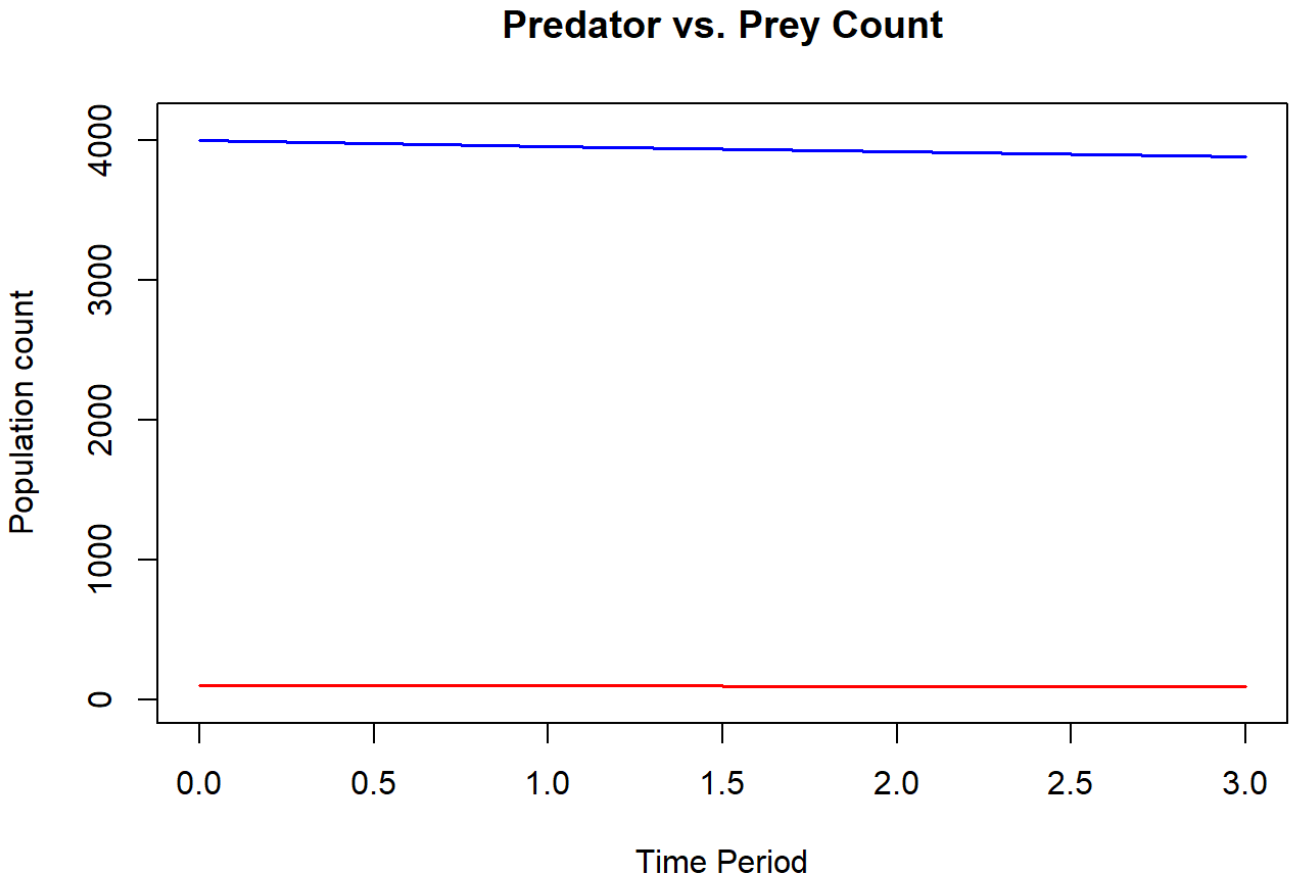
The computing time for all of the calculations is very small, so it's hard to compare. But I would expect that using the vectorization method as in part (d) would be faster than using loops, and it is for our only large computation (x = 1.01, n = 5000).

2)

```r
#first an iterative function
lotka.volt <- function(x, y, bx, by, dx, dy){
  xt = x
  yt = y
  time = 0
  mat = matrix(ncol = 3 , nrow = 1)
  mat[1,]<- c(time, x, y)
  while(xt > 3900){
    time = time + 1
    xt = x + bx*x - dx*x*y
    yt = y + by*dx*x*y - dy*y
    x = xt
    y = yt
    mat <- rbind(mat, c(time, x, y))
  }
  lst <- list(mat[,1], mat[,2], mat[,3])
  names(lst) <- c("time", "prey", "predator")
  return(lst)
}

#run the function using specified variables
lv <- lotka.volt(4000, 100, .04, .1, .0005, .2)
#the prey don't last too long
lv
```

```
## $time
## [1] 0 1 2 3
##
## $prey
## [1] 4000.000 3960.000 3920.400 3881.588
##
## $predator
## [1] 100.0000 100.0000  99.8000  99.4028
```

```r
#plot the data
attach(lv)
plot(x = time, y = predator, type = "l", lwd = 1.5, lty = 1, col = "red", xlab = "
Time Period", ylab = "Population count",
     main = "Predator vs. Prey Count", ylim = c(0,max(prey)+ 100))
lines(x = time, y = prey, lwd = 1.5, lty = 1, col = "blue")
```



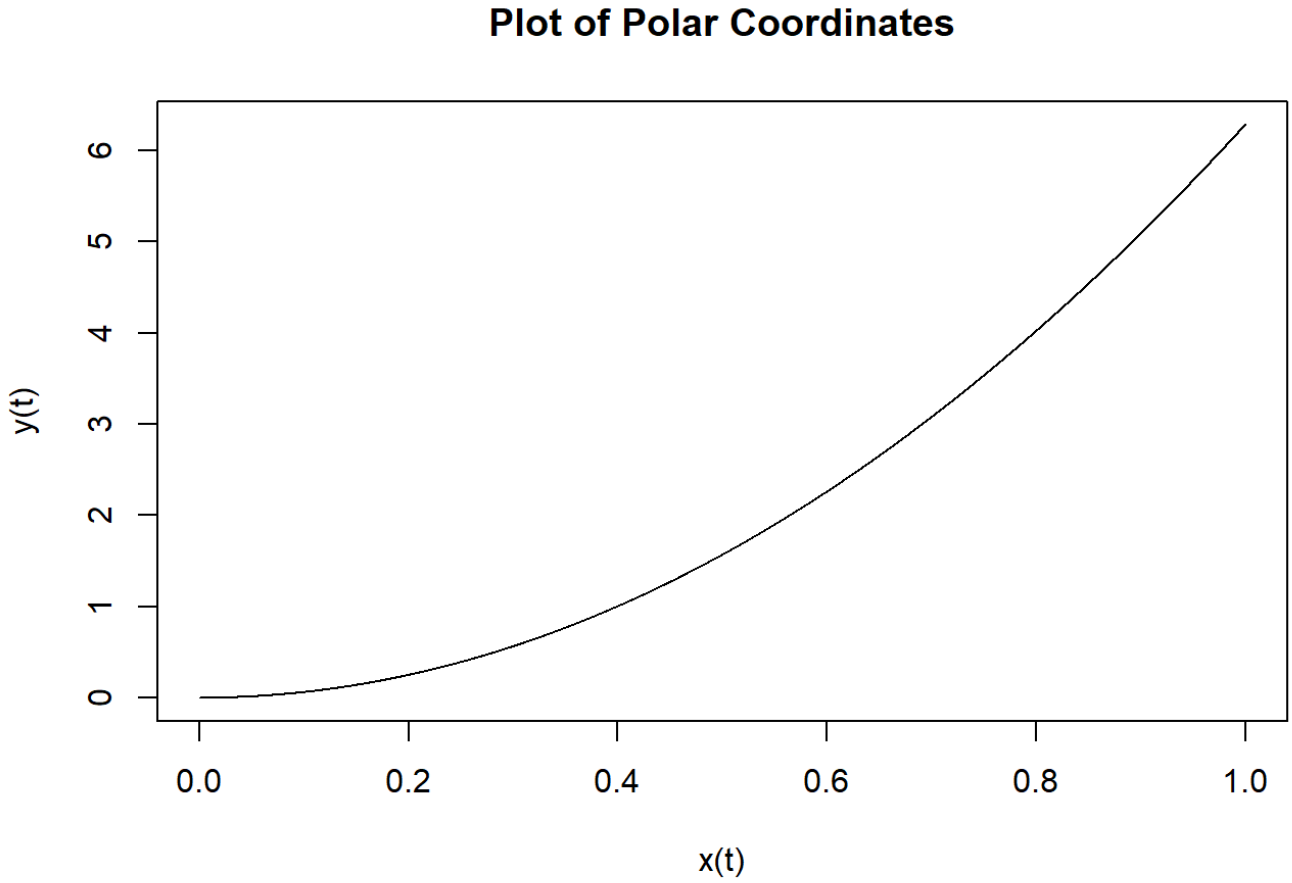
3)

```r
#Write the function using the rules of the game
craps <- function(){
  roll <- sample(6, size = 2, replace = TRUE)
  sum.r <- sum(roll)
  sum.s <- 0
  outcome <- "Loss"
  nroll <- 1
  if (sum.r == 7 || sum.r == 11) outcome = "Win" else
    while (sum.s != sum.r){
      nroll <- nroll + 1
      roll <- sample(6, size = 2, replace = TRUE)
      sum.s <- sum(roll)
      if (sum.s == 7 || sum.s == 11) break
      if (sum.s == sum.r) outcome = "Win"
    }
  out <- list(sum.r, sum.s, nroll, outcome)
  names(out) <- c("First Roll", "Last Roll", "Nrolls","Outcome")
  return(out)
}

#Returns a list with the first roll, the last roll, total number of rolls, and the
outcome
craps()
```

```
## $`First Roll`
## [1] 12
##
## $`Last Roll`
## [1] 11
##
## $Nrolls
## [1] 2
##
## $Outcome
## [1] "Loss"
```

4)

```
polarize <- function(start, end, by = .001){
  i<- start
  s <- matrix(ncol = 2, nrow = 1)
  s[1,] <- c(sqrt(i),2*pi*i)
  while(i < end){
    s <- rbind(s, c(sqrt(i),2*pi*i))
    i <- i + by
  }
  plot(x = s[,1], y = s[,2], type="l", xlab = "x(t)", ylab = "y(t)", main = "Plot
of Polar Coordinates",
       xlim = c(min(s[,1]), max(s[,1])), ylim = c(min(s[,2]), max(s[,2])))
}

#Let's see the result for t from 0 to 1
polarize(0 , 1, by=.0001)
```



5)

```
#start with this matrix of random numbers and another of variances
x <- matrix(rnorm(n = 500), ncol = 5)
varx <- var(x)
#Compute square roots of variances for each element
sds <- sqrt(diag(varx))
#Sweep it out by row
swept1 <- sweep(varx, MARGIN = 1, sds, FUN = "/")
#Then by column to get the correlation matrix
r <- sweep(swept1 , MARGIN = 2, sds, FUN = "/")
r
```

```
##              [,1]        [,2]        [,3]        [,4]        [,5]
## [1,]  1.000000000 -0.15056562  0.13379731 0.003010352 -0.04519766
## [2,] -0.150565622  1.00000000 -0.16127719 0.121501740 -0.08600153
## [3,]  0.133797306 -0.16127719  1.00000000 0.070338610  0.06489507
## [4,]  0.003010352  0.12150174  0.07033861 1.000000000  0.02542594
## [5,] -0.045197659 -0.08600153  0.06489507 0.025425939  1.00000000
```

```
#We can compare it to R's built in function
#Should have the same result
cor(x)
```

```
##              [,1]        [,2]        [,3]        [,4]        [,5]
## [1,]  1.000000000 -0.15056562  0.13379731 0.003010352 -0.04519766
## [2,] -0.150565622  1.00000000 -0.16127719 0.121501740 -0.08600153
## [3,]  0.133797306 -0.16127719  1.00000000 0.070338610  0.06489507
## [4,]  0.003010352  0.12150174  0.07033861 1.000000000  0.02542594
## [5,] -0.045197659 -0.08600153  0.06489507 0.025425939  1.00000000
```

6)

a)

i)

```
#read in the data
#need to read in the array correctly
meas <- aperm(array(scan(file = "etcpod_05-400_102307_1_trig01.dat"), dim = c(72,72, 150)), perm =c(2,1,3))
```

ii)

A)

```
#first collect the first 5 images
first5 <- meas[,,1:5]
#then average them
f5.avg <- apply(first5, MARGIN = c(1,2), FUN = mean)
dim(f5.avg)
```

```
## [1] 72 72
```

B)

```
#b
#We can sweep out each matrix in the array using the averages calculated above
bg.rm <- sweep(meas, MARGIN = c(1,2), f5.avg, FUN = "-")
dim(bg.rm)
```

```
## [1]  72  72 150
```

C)

```
#I find the hottest pixel overall, then use the arrayIND function to find where it's located
hottest.index <- arrayInd(which.max(bg.rm), dim(bg.rm))
hottest.index
```

```
##      [,1] [,2] [,3]
## [1,]   45   40   93
```

```
#The hottest pixel is in frame 93
hottest.index[1,3]
```

```
## [1] 93
```

D)

```
#Create a function that we can apply to each frame
smooth.it <- function(m){
  oldmat <- m
  #Create a new frame by expanding the old one on each edge
  #I am just going to add a box of NAs around the image, then ignore them
  m.1 <- rbind(c(rep(NA, times = ncol(m))), m, c(rep(NA, times = ncol(m))))
  newmat <- cbind(c(rep(NA, times = nrow(m)+2)), m.1, c(rep(NA, times = ncol(m)+2)
))
  #Now try to smooth it out
  for(i in 1:ncol(m)){
    for(j in 1:nrow(m)){
      oldmat[i,j] <- mean(c(newmat[i, j+1], newmat[i+1, j], newmat[i+1, j+1],
                            newmat[i+1, j+2], newmat[i+2, j+1]), na.rm = TRUE)
    }
  }
  return(oldmat)
}

#apply the function to our array of frames
smoothed.1 <- apply(bg.rm, MARGIN = 3, FUN = smooth.it)
#fit our new frames into an array
smoothed <- array(smoothed.1, dim=c(72,72,150))
dim(smoothed)
```

```
## [1]  72  72 150
```

```
#Now we find the frame with the hottest pixel again, just as above
#I find the hottest pixel overall, then use the arrayIND function to find where it'
s located
hottest.sm <- arrayInd(which.max(smoothed), dim(smoothed))
hottest.sm
```

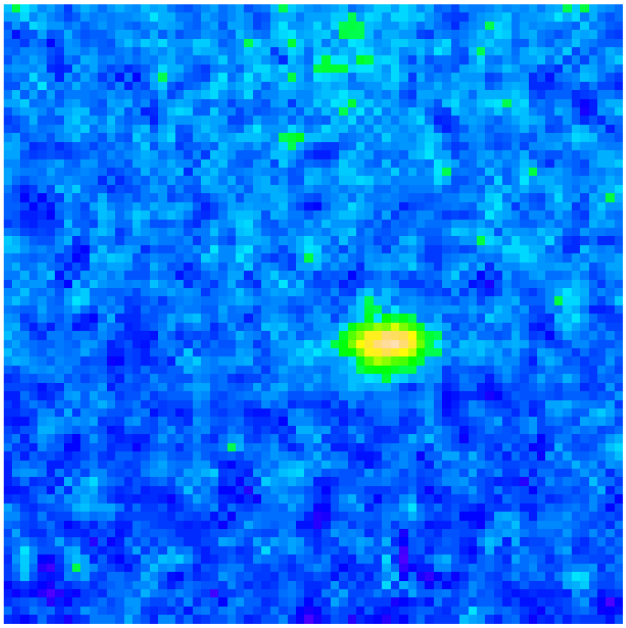
```
##      [,1] [,2] [,3]
## [1,]   46   40   93
```

```
#The hottest pixel is still in frame 93
hottest.sm[1,3]
```

```
## [1] 93
```

E)

```
#reverse the rows so that image reads in correctly
#We know that the hot spot should be around (40,45) in the frame, right and below t
he center (36,36)
smoothed.rev<- smoothed[,72:1,]
#a plot of our hottest frame shows where the hot spot is
image(z=smoothed.rev[,,93], col = topo.colors(72), axes = F, asp = 1)
```



b)

i)

```
require(readxl)
```

```
## Loading required package: readxl
```

```
## Warning: package 'readxl' was built under R version 3.2.5
```

```
#read in the data set
#I removed the quaternion columns and blank columns before reading in the data
prelim <- as.data.frame(read_excel(path = "PreliminaryData.xlsx"))
head(prelim)
```

```
##    x y Grain ID DD (old method) DD (L1 method)
## 1  0 0        1        16.08350       14.32580
## 2 10 0        1        16.36391       14.40436
## 3 20 0        1        15.95513       13.90034
## 4 30 0        1        16.12734       14.38407
## 5 40 0        1        16.00242       14.66501
## 6 50 0        1        16.29435       14.72672
```

ii)

The first code is the easy way, using tapply() and checking to make sure the results are the same

```
#First, verify the y column is sorted
is.unsorted(prelim$y)
```

```
## [1] FALSE
```

```
#Hope to get the same number for each resulting length of unique values (496)
#check to see if each x value is sorted within each y
check.sort <- tapply(X = prelim$x, INDEX = as.factor(prelim$y), FUN= is.unsorted)
unique(check.sort) #All of them are sorted
```

```
##     0
## FALSE
```

```
#check to see if each y value has the same number of corresponding x values
check.length <- tapply(X = prelim$x, INDEX = as.factor(prelim$y), FUN= length)
unique(check.length) #we should have 498 x values for each y
```

```
##   0
## 498
```

```
#Check to make sure all of the x values are unique within each y value
check.unique <- length(tapply(X = prelim$x, INDEX = as.factor(prelim$y), FUN= uniq
ue))
check.unique
```

```
## [1] 496
```

```
#Check mins and maxes to make sure they're all the same
check.min <- tapply(X = prelim$x, INDEX = as.factor(prelim$y), FUN= min)
unique(check.min)
```

```
## 0
## 0
```

```
length(check.min)
```

```
## [1] 496
```

```
check.max <- tapply(X = prelim$x, INDEX = as.factor(prelim$y), FUN= max)
unique(check.max)
```

```
##    0
## 4970
```

```
length(check.max)
```

```
## [1] 496
```

All of the above checks out, so I conclude our data is sorted and not missing any values. The below code is a much more difficult way of doing it with for loops, but I will not compile it here because it takes too long. But it does work if you want to try.

```
#Check to see if all possible measurements are present
#All increments are in 10s, so we can just run a nested for loop to check
#I make a matrix of missing values and also count the number of times the loop runs
to make sure it's right
check <- function(z){
  i <- 0; ct <- 1;
  missing.vals <- matrix(0, nrow = 1, ncol = 2)
  available.vals <- matrix(0, nrow = 1, ncol = 2)
  #hold y values constant on the outside loop, test for each x inside the loop
  for(i in seq(0,max(z[,2]), by = 10)){
    j<-0
    for(j in seq(0,max(z[,1]), by = 10)){
      if(!(z[ct,2] == i && z[ct, 1] == j)) missing.vals = rbind(missing.vals, c(j
, i)) else available.vals = rbind(available.vals, c(j,i))
      j <- j+ 10
      ct <- ct + 1
    }
    i<- i+10
  }
  list.f <- list(missing.vals, available.vals, ct)
  names(list.f) <- c("missing", "avail", "count")
  return(list.f)
}

#Put all of our x values in to a matrix to test if we have all of them
testmat <- cbind(prelim[,1:2])

#run our function to test
testted <- check(testmat)
#check some of the output
dim(testted$avail)
testted$missing
testted$count
```

The matrix of missing values is empty, and we've gone through all of the indices, so it's clear that the data is sorted and there is nothing missing. All of our measurements are available (but some are

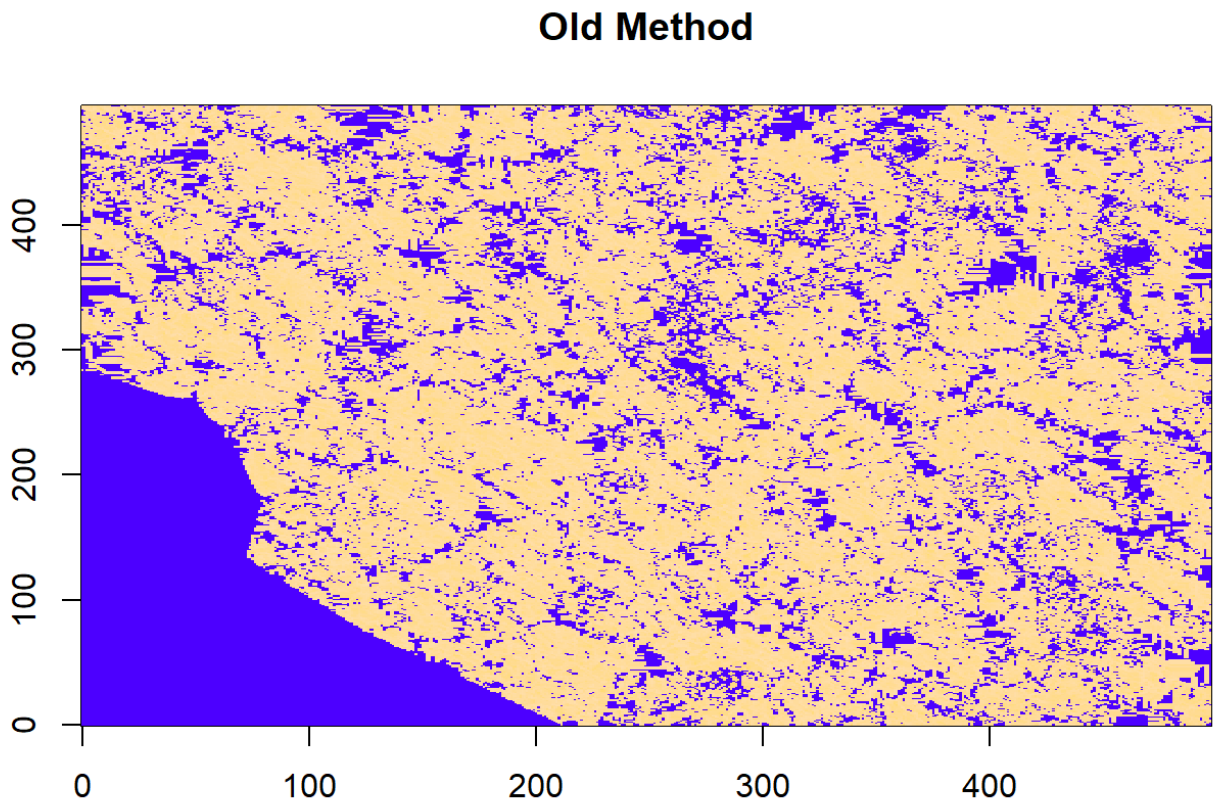
0s).

iii)

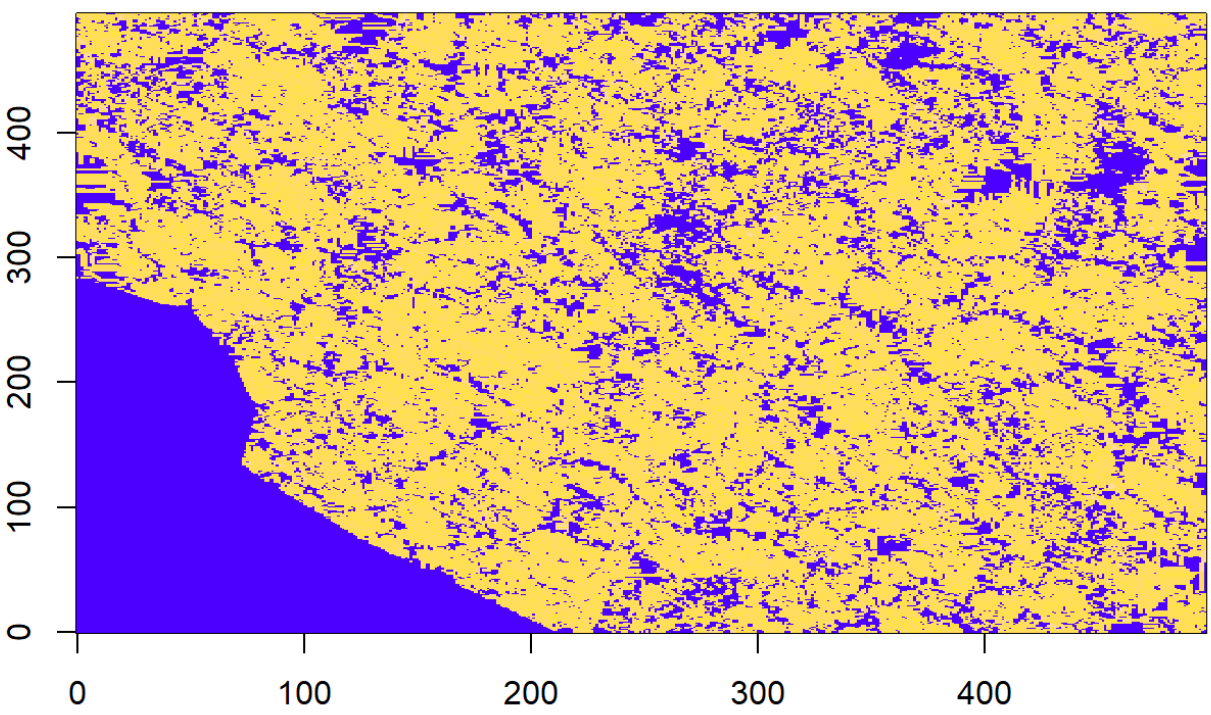
```
#iii
#We already confirmed that they were in order, so we can just fill the matrix by co
lumns
mat.old <- matrix(prelim$`DD (old method)`, ncol = max(prelim$y)/10 + 1, nrow = ma
x(prelim$x)/10 + 1)
mat.new <- matrix(prelim$`DD (L1 method)`, ncol = max(prelim$y)/10 + 1, nrow = max
(prelim$x)/10 + 1)
```

iv)

```
#plot the images, with row order reversed so that image reads them correctly from t
op to bottom
#The second image will have a few more visible "fractures" for the new method
image(z=mat.old[,496:1], x = c(0:497), y = c(0:495), col = topo.colors(120), main
= "Old Method", xlab = "", ylab = "")
```




```
image(z=mat.new[,496:1], x = c(0:497), y = c(0:495), col = topo.colors(120), main
= "New Method", xlab = "", ylab = "")
```


New Method
0
100
200
300
400
0
100
200
300
400