Lecture 4
Matrix

GEOG 489

SPRING 2020

Quiz 1

1) Write a function to count the number of values in a vector that can be divided by 3

2) Write a function to print the positions of values in a vector that can be divided by 3

Testing a function

```
threecount <- function(x) {
    k <- 0 # assign 0 to k
    for (n in x) {
        if(n %% 3 == 0)
        {
            k <- k+1 # %% is the modulo operator
        }
    }
    return(k)
}
```

Testing a function

```
# Rather than try to test the whole function,  we will "manually" assign the parameter and test the internals of the code:

x <- seq(1:3) # This is the parameter

# Now paste in the internals of the function  EXCEPT the return statement, and include some print statements:

k <- 0 # assign 0 to k

for (n in x) {
	if(n %% 3 == 0)
	{
		print(paste(n,“can be divided by 3"))
		k <- k+1 # %% is the modulo operator
	}
}

# Run this line-by-line, then test k:

k
```

Quiz 1

2) Write a function to print the positions of values in a vector that can be divided by 3

```
printposition <- function(x){
  print(which(x %% 3 == 0))
}
testdata <- c(1,3,5)
printposition(testdata)
```

Matrix

1) Create matrix

Matrices are created in "column-major order", so all of column 1 is stored first, then column 2, etc.

y <- matrix(data=c(1,2,3,4),ncol=2,nrow=2)

y <- matrix(data=c(1,2,3,4),2,nrow=2)

2) Matrix indexing

y[,2] # shows us everything in column 2 (notice it prints out in vector format)

y[-2,]

We can perform assignments using row indices as well:

y<-matrix(1:6,nrow=3,ncol=2)

y[c(1,3),] <- matrix(c(1,1,8,12),nrow=2)

Matrix

3) Define an empty matrix

```
y <- matrix(nrow=2,ncol=2)

y # As with empty vectors, an empty matrix defaults to mode(y) == logical,  but will switch to the mode of the first element we assign.

y[1,1] <- 1

y[2,1] <- 2

y[1,2] <- 3

y[2,2] <- 4

# Notice that we CAN fill in a matrix row-by-row (like an image) by using the byrow=TRUE parameter:

m <- matrix(1:6,nrow=2,byrow=TRUE)
```

Matrix

4) Matrix operation

Basic (and advanced) linear algebra operations are available in R

```
y <- matrix(c(1,2,3,4),nrow=2,ncol=2)

# Matrix multiplication (refer to your linear algebra textbooks for the description):

y %*% y

# Element-wise multiplication:

y*y

# Element-wise addition:

y+y
```

Install Packages

1) install packages (install our first "add-on" package for R)

Intall the "raster" package

install.packages("raster") # In this case, quotes are required.

2) load packages

library("raster")

Matrix and raster file

Matrix notation allows us to interact with raster file

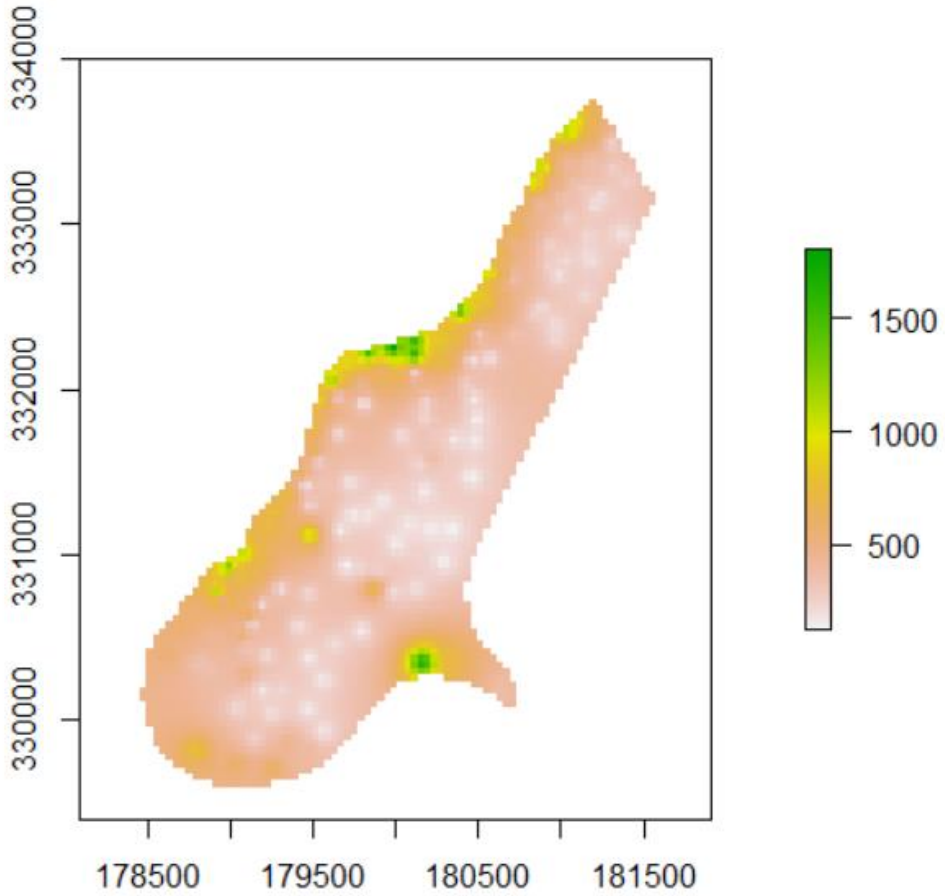
raster(test.grd)

```
class       : RasterLayer
dimensions  : 115, 80, 9200  (nrow, ncol, ncell)
resolution  : 40, 40  (x, y)
extent      : 178400, 181600, 329400, 334000  (xmin, xmax, ymin, ymax)
coord. ref. : +init=epsg:28992 +towgs84=565.237,50.0087,465.658,-0.406857,0.350733,-1.87035
lon_0=5.38763888888889 +k=0.9999079 +x_0=155000 +y_0=463000 +ellps=bessel +units=m +no_defs
data source : C:\Users\chunyuan\Documents\R\win-library\3.4\raster\external\test.grd
names       : test
values      : 128.434, 1805.78  (min, max)
```

Matrix and raster file

Matrix notation allows us to interact with raster file

plot()

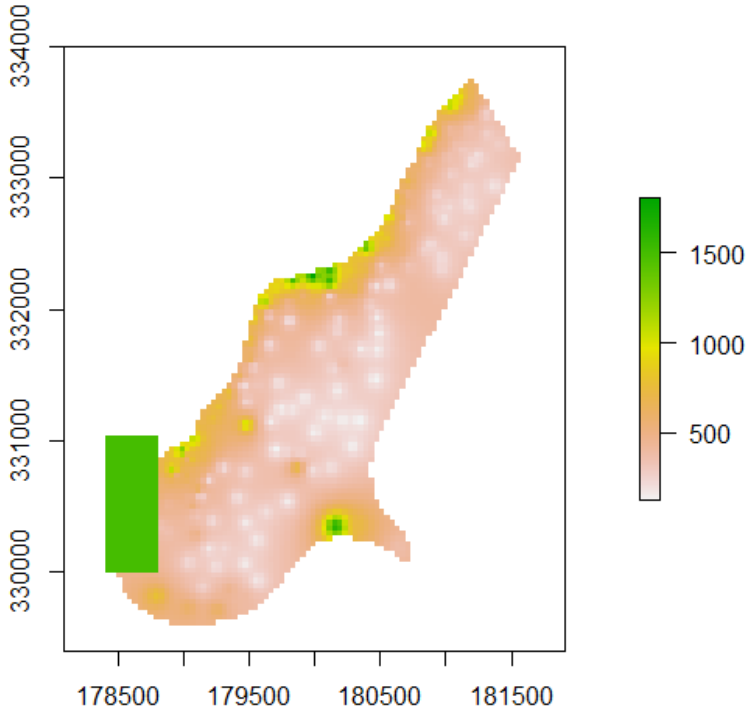


Matrix and raster file

Rasters are basically matrices, and use some of the same basic notation.

So, for instance, we can re-assign certain values:

```
r[75:100,1:10] <- 1500
plot(r)
```



Matrix

5) Filtering on matrices

```
x <- matrix(c(1,2,3,2,3,4),nrow=3,ncol=2)

# Only returns rows where the second column entry is >= 3

x[x[,2] >= 3,]


m <- matrix(1:6,nrow=3)

m[m[,1] > 1 & m[,2] > 5,]
```

Matrix

6) apply() function: Apply is a very powerful tool for use with matrices. It allows an easy way to apply a function on a row-by-row, or column-by-column basis.

apply(X,MARGIN,FUN,...)

Where:

X is a matrix or array

MARGIN is the dimension to use (1 = row by row, 2 = column by column)

FUN is a function to be applied

Matrix

6) apply() function: Apply is a very powerful tool for use with matrices. It allows an easy way to apply a function on a row-by-row, or column-by-column basis.

apply(X,MARGIN,FUN,...)

```
# Calculate the mean value for each column of a matrix

z <- matrix(1:6,nrow=3)

apply(X=z,MARGIN=2,FUN=mean)
```

Matrix

6) apply() function: Apply is a very powerful tool for use with matrices. It allows an easy way to apply a function on a row-by-row, or column-by-column basis.

```
# We can define our own function:

z <- matrix(1:6,nrow=3)

f <- function(x) { x/c(2,8) }

y <- apply(X=z,MARGIN=1,FUN=f)
```

Matrix

7) Add/delete matrix rows and columns:

```
ones = rep(1,4)

z <- matrix(seq(10,160,by=10),nrow=4,ncol=4)


rbind(z,ones)

cbind(ones,z)
```

Matrix

8) Convert a vector to a matrix

```
u <- 1:3 # A vector

v <- as.matrix(u) # Now a matrix

attributes(u)

attributes(v)
```

Matrix

9) Higher dimensional arrays

We can create arrays of arbitrary dimensions in R. For example, we have three dimensions representing the x location, y location, and layer.

```
# Say we have two layers as matrices:

layer1 <- matrix(1:6,nrow=3)

layer2 <- matrix(101:106,nrow=3)

merged_array <- array(data=c(layer1,layer2),dim=c(3,2,2))

# subset an array
merged_array[3,2,1]
```

Assignment 1

Your goal is to write a function that takes two inputs:

x = a vector of numbers

d = a single value

The function should return a vector of numeric indices of all vector locations in which the element of x divided by d has no remainder.

Assignment 1

There are a few requirements of the code:

1) the function name should be "noRemainderIndices"

2) x should have no default, but d should default to 2.

3) the code should stop with a warning message if d is anything but a single element of mode "numeric"

4) if no indices are found, the code should return NULL

5) comment your code in at least two places

6) the code should be submitted as a R file

7) At the top of your code, please include a comment section with your name and assignment number

Assignment 1

Test case:

```
noRemainderIndices(x=2:12,d=3)
# [1]  2  5  8 11
noRemainderIndices(x=2:12) # d should default to 2
# [1]  1  3  5  7  9 11
noRemainderIndices(x=2:12,d=13) # No remainders, so should return NULL
# NULL
noRemainderIndices(x=2:12,d=1:3) # d does not have a length of 1, so returns an error
# Error in noRemainderIndices(x = 2:12, d = 1:3) :
#d must have a length of 1
noRemainderIndices(x=2:12,d="abc") # d isn't numeric, so returns and error
# Error in noRemainderIndices(x = 2:12, d = "abc") : d must be numeric
```

Assignment 1 is due on Tuesday, Feb. 11 at midnight.

Please submit your assignment on Compass 2g