3. Base R - Functionals and Matrices

CT5102 - J. Duggan

Additional Function Features

- Functions
  - ... argument
  - do.call() function
  - Replacement functions
  - returns a result (Matloff 2009).
- Functionals
  - sapply()
  - lapply()
  - apply()

... argument

- There is a special argument called ...
- This argument will match any arguments not otherwise matched, and can be easily passed on to other functions

```
mysum <- function(v,...){
  sum(v,...)
}

x <- c(1,2,3, NA)
mysum(x)
```

```
## [1] NA
```

```
mysum(x,na.rm=T)
```

```
## [1] 6
```

do.call() function

- Calling a function, given a list of arguments

```
args <- list(c(1:3),NA,na.rm=T)
args
```

```
## [[1]]
## [1] 1 2 3
##
## [[2]]
## [1] NA
##
## $na.rm
## [1] TRUE
```

```
do.call(sum, args)
```

```
## [1] 6
```

Replacement Functions

- Replacement functions act like they modify their arguments in place, and have the special name `xxx<-`
- They typically have two arguments (x and value), although they can have more, and they must return the modified object
- The new value must be passed as a parameter named **value**

dim() function

```
x <- 1:6
x
```

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

```
dim(x) <- c(2,3)
x
```

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

Replacement Function Example

```
`second<-` <- function(x, value){
  x[2] <- value
  x
}

x <- 1:5
second(x) <- 78
x
```

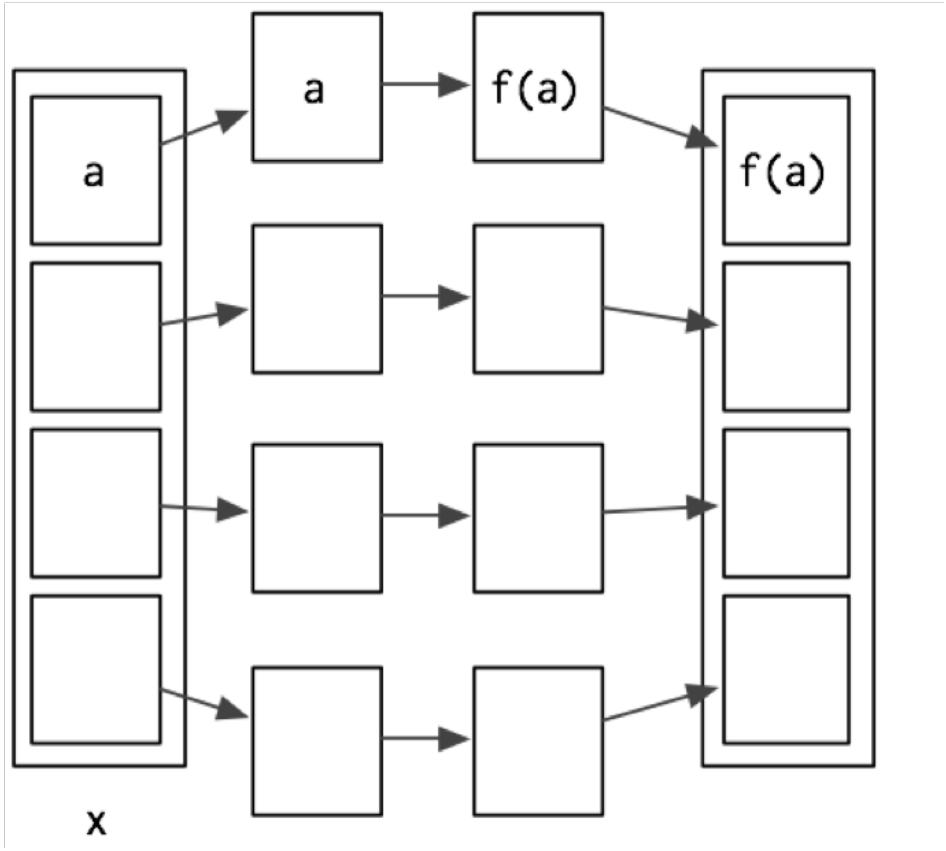
```
## [1]  1 78  3  4  5
```

Functionals

- A functional is a function that takes a function as an input and returns a vector as output
- Commonly used as an alternative for loops
- Common ones
  - sapply()
  - apply()
  - lapply()

Common Pattern (Wickham 2015)

- Create a container for output
- Apply f() to each component of the list
- Fill the container with the results



my_sapply()

```
my_sapply <- function(x,f,...){
  out <- vector("list",length(x))
  for (i in seq_along(x)){
    out[[i]] <- f(x[[i]],...)
  }
  unlist(out)
}

my_sapply(1:5,
          function(v)v*2-10)
```

```
## [1] -8 -6 -4 -2  0
```

sapply()

- The general form of the **sapply(x,f,fargs)** function is as follows:
  - **x** is the target vector or list
  - **f** is the function to be called and applied to each element
  - **fargs** are the optional set of arguments that can be applied to the function f.
- sapply() returns a vector

```
x <- 1:3
y <- sapply(x,function(v)v*2)
y
```

```
## [1] 2 4 6
```

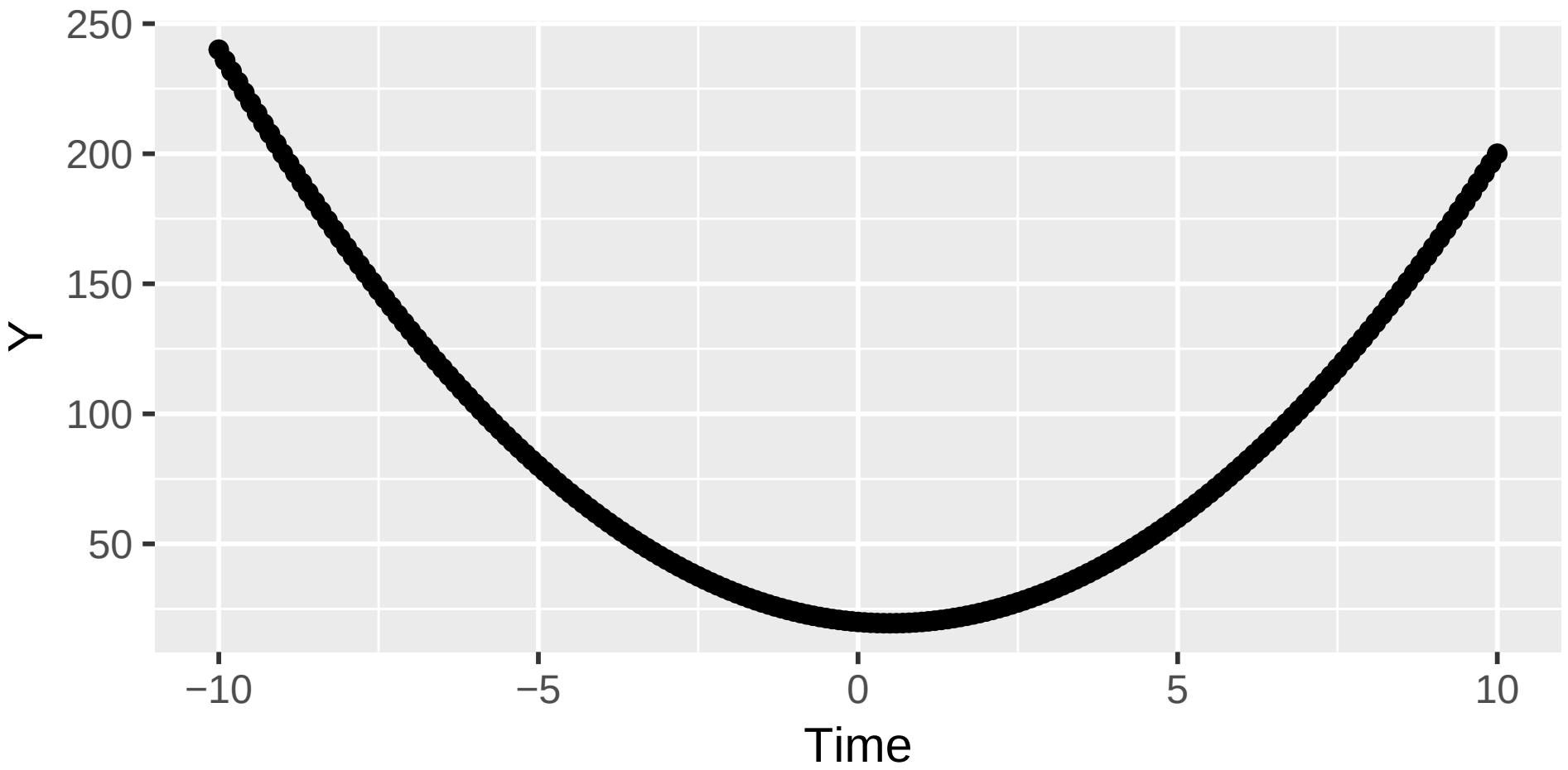
sapply() - processing a quadratic equation

```
time <- seq(-10,10,by=0.1)
y <- sapply(time,function(x,a,b,c)a*x^2+b*x+c,a=2,b=-2,c=20)
m <- matrix(nrow=length(time),ncol=2)
colnames(m)<- c("Time","Y")
m[,1] <- time
m[,2] <- y
m[1:5,]
```

```
##        Time      Y
## [1,] -10.0 240.00
## [2,]  -9.9 235.82
## [3,]  -9.8 231.68
## [4,]  -9.7 227.58
## [5,]  -9.6 223.52
```

Plotting the results

```
ggplot(data=data.frame(m),aes(x=Time,y=Y))+
        geom_point()
```



Matrices

	Homogenous	Heterogenous
1d	Atomic Vector	List
2d	**Matrix**	Data Frame/Tibble
nd	Array	

- A matrix can be initialized from a vector, where the numbers of rows and columns are specified.
- R stores matrices by column-major order, and by default matrices are filled in this manner.

Declaring a matrix

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

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

```
dim(a)
```

```
## [1] 2 3
```

Adding rows through rbind()

```
a
```

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

```
a1 <- rbind(a,7:9)
a1
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
## [3,]    7    8    9
```

```
dim(a1)
```

```
## [1] 3 3
```

Adding columns through cbind()

```
a
```

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

```
a2 <- cbind(a,7:8)
a2
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    3    5    7
## [2,]    2    4    6    8
```

```
dim(a2)
```

```
## [1] 2 4
```

Naming rows and columns

```
rownames(a) <- c("R1","R2")
a
```

```
##    [,1] [,2] [,3]
## R1    1    3    5
## R2    2    4    6
```

```
colnames(a) <- c("C1","C2","C3")
a
```

```
##    C1 C2 C3
## R1  1  3  5
## R2  2  4  6
```

Subsetting Matrices

- The most common way of subsetting 2d matrix is a simple generalisation of 1d subsetting
- Supply a 1d index for each dimension, separated by a comma
- Blank subsetting is useful, as it lets you keep all rows or all columns

```
b <- matrix(1:9,ncol=3,nrow=3)
rownames(b) <- c("R1","R2","R3") # optional
colnames(b) <- c("C1","C2","C3") # optional
b
```

```
##    C1 C2 C3
## R1  1  4  7
## R2  2  5  8
## R3  3  6  9
```

Subsetting by row index - 1/2

```
b
```

```
##    C1 C2 C3
## R1  1  4  7
## R2  2  5  8
## R3  3  6  9
```

```
b[1:2,]
```

```
##    C1 C2 C3
## R1  1  4  7
## R2  2  5  8
```

```
b[c("R1","R2"),]
```

```
##    C1 C2 C3
## R1  1  4  7
## R2  2  5  8
```

Subsetting by row index - 2/2

```
b
```

```
##    C1 C2 C3
## R1  1  4  7
## R2  2  5  8
## R3  3  6  9
```

```
b[c(T,T,F),]
```

```
##    C1 C2 C3
## R1  1  4  7
## R2  2  5  8
```

```
b[-c(1,2),]
```

```
## C1 C2 C3
##  3  6  9
```

Subsetting by column index - 1/2

```
b
```

```
##    C1 C2 C3
## R1  1  4  7
## R2  2  5  8
## R3  3  6  9
```

```
b[,1:2]
```

```
##    C1 C2
## R1  1  4
## R2  2  5
## R3  3  6
```

```
b[,c("C1","C3")]
```

```
##    C1 C3
## R1  1  7
```

Subsetting by column index - 2/2

```
b
```

```
##    C1 C2 C3
## R1  1  4  7
## R2  2  5  8
## R3  3  6  9
```

```
b[1:2,c(T,T,F)]
```

```
##    C1 C2
## R1  1  4
## R2  2  5
```

```
b[1:2,-c(1,3)]
```

```
## R1 R2
##  4  5
```

Matrix Operators/Functions

Operator	Description
A * B	Element-wise multiplication
A / B	Element-wise division
A %*% B	Matrix Multiplication
t(A)	Transpose of A
eigen(A)	List of eigenvalues/eigenvectors for A

apply() - process matrices/data frames

The general form of this function is **apply(m, dimcode, f, fargs)**, where: - m is the target matrix - dimcode identifies whether it's a row or column target. The number 1 applies to rows, whereas 2 applies to columns - f is the function to be called, and fargs are the optional set of arguments that can be applied to the function f.

```
m <- matrix(1:10,nrow = 2)
m
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5    7    9
## [2,]    2    4    6    8   10
```

```
apply(m,1,sum) # sum the row
```

```
## [1] 25 30
```

```
apply(m,2,sum) # sum the columns
```

Challenge 3.1

- Create a 10x10 matrix to represent connections between people on social media (random seed=100)
- Label the people [A..J], with named rows and columns.
- Randomly populate the matrix with 1s and 0s. The number 1 means someone follows/is followed by another person. Ensure that all diagonals are 0 (you should use an appropriate R matrix operation for this).
- Each row contains information on the people a person follows.
- Each column contains information on who follows a person.

Challenge 3.2

- Create a matrix of grades for subjects (50 students x 5 subjects)
- Each column is a different subject, with grades drawn from a random normal distribution (function rnorm). - Name the columns
- Each row is a students result
- Use apply to calculate the average mark for each student, and add this result as a new column to the matrix

Summary - Functionals and Matrices

- Functionals are functions that takes a function as an input and returns a vector as output (can be used as a looping structure)
  - The apply family in R are functionals (**apply**, **sapply**, **lapply**)
  - The package **purrr** is now being used instead of **apply**
- Matrix is used for 2-d homogenous data. Filtering rules similar to atomic vectors. Matric operations are (by default) element-wise, and matrix multiplication also supported.