

# GOVT PhD Math Camp Coding Lab Day 3

Theodore Landsman

Georgetown University

August 18 2021

# Commands and Functions

- We use a command/function to perform some tasks on an object/objects

# Commands and Functions

- We use a command/function to perform some tasks on an object/objects
- Argument: the definitions, directions, or objects that are passed to a command/function

# Commands and Functions

- We use a command/function to perform some tasks on an object/objects
- Argument: the definitions, directions, or objects that are passed to a command/function
- When we specify multiple arguments,
  - separate the arguments by commas
  - it is desirable to specify them along with their names unless they are obvious
  - The code looks like `funcname(arg1 = input1, arg2 = input2)`

# Commands and Functions Example

```
log(num.2)
```

```
## [1] 1.3862944 1.7917595 1.6094379 0.6931472 1.0986123
```

```
sqrt(num.2)
```

```
## [1] 2.000000 2.449490 2.236068 1.414214 1.732051
```

```
length(num.1)
```

```
## [1] 9
```

```
sum(num.1)
```

```
## [1] 13
```

# Commands and Functions Example Continued

```
sort(num.1)
```

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

```
sort(num.1, decreasing = TRUE)
```

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

- We can access help files for functions by typing either `?funcname` or `help("funcname")` into the console.

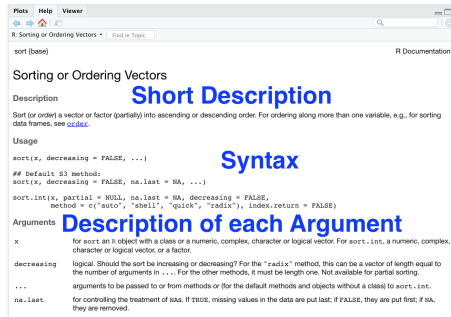
# R Help Documentation

- We can access help files for functions by typing either `?funcname` or `help("funcname")` into the console.
- R help files are included for all base R functions *and* all package functions, coders are required to provide help documentation as part of uploading their packages to CRAN.



# R Help Documentation Continued

## • Output of `help("sort")`



The screenshot shows the top portion of the R help file for the 'sort' function. It includes the title 'Sorting or Ordering Vectors', a 'Description' section explaining that 'sort' orders a vector or factor (partially) into ascending or descending order, and an 'Usage' section showing the basic syntax: `sort(x, decreasing = FALSE, ...)`. The 'Arguments' section begins, defining 'x' as the object to be sorted and 'decreasing' as a logical flag. The text is partially obscured by blue annotations: 'Short Description' over the description, 'Syntax' over the usage, and 'Description of each Argument' over the arguments section.

```
sort (base)

Sorting or Ordering Vectors

Description
Sort (or order) a vector or factor (partially) into ascending or descending order. For ordering along more than one variable, e.g., for sorting data frames, see order.

Usage
sort(x, decreasing = FALSE, ...)

## Default S3 method:
sort(x, decreasing = FALSE, na.last = NA, ...)

sort.int(x, partial = NULL, na.last = NA, decreasing = FALSE,
method = c("auto", "shell", "quick", "radix"), index.return = FALSE)

Arguments
x          for sort an R object with a class or a numeric, complex, character or logical vector. For sort.int, a numeric, complex, character or logical vector, or a factor.
decreasing logical. Should the sort be increasing or decreasing? For the "radix" method, this can be a vector of length equal to the number of arguments in ... For the other methods, it must be length one. Not available for partial sorting.
...        arguments to be passed to or from methods or (for the default methods and objects without a class) to sort.int.
na.last    for controlling the treatment of NAs. If TRUE, missing values in the data are put last; if FALSE, they are put first; if NA, they are removed.
```

(a) Top of help file

## References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988). *The New S Language*. Wadsworth & Brooks/Cole.

Knuth, D. E. (1998). *The Art of Computer Programming, Volume 3: Sorting and Searching*, 2nd ed. Addison-Wesley.

Sedgewick, R. (1998). A new upper bound for Shellsort. *Journal of Algorithms*, **7**, 159–173. doi: [10.1016/0196-6774\(98\)00001-5](https://doi.org/10.1016/0196-6774(98)00001-5).

Singleton, R. C. (1969). Algorithm 347: an efficient algorithm for sorting with minimal storage. *Communications of the ACM*, **12**, 185–186. doi: [10.1145/362875.362901](https://doi.org/10.1145/362875.362901).

## See Also

['Comparison'](#) for how character strings are collated.

[order](#) for sorting on or reordering multiple variables.

[is.unsorted.rang](#).

## Executable Examples of How to Use it

```
require(stats)

x <- rswiss$Education[1:25]
x[ sort(x), partial = c(10, 15) ]

## illustrate 'stable' sorting (of ties):
sort(c(10:13, 2:12), method = "shell", index.return = TRUE) # is stable
## $x : 2 3 3 4 4 5 5 6 6 7 7 8 8 9 9 10 10 11 12
## $ix: 9 8 10 7 13 6 12 5 13 4 14 3 15 2 16 1 17 18 19
sort(c(10:13, 2:12), method = "quick", index.return = TRUE) # is not
```

(b) Bottom of help file

- Don't understand what the help files are saying? Google search, ask others (including me!)

# Function Exercises

- Calculate  $10!$  and  $\sqrt{16}$  *without* functions.  
Then do so using the functions `fact()` and `sqrt()`
- For the `num.1` object we created earlier,
  - Calculate the product of all the numbers.
  - Count the number of elements larger than 0.
- Execute all five examples of `sum` under `help("sum")`

# Creating Vectors and Matrices

- As we covered yesterday, in R we create vectors using `c()` command.

# Creating Vectors and Matrices

- As we covered yesterday, in R we create vectors using `c()` command.
- We can create matrices with `matrix()` command

## Usage

```
matrix(data, nrow, ncol, byrow)
```

# Creating Vectors and Matrices

- As we covered yesterday, in R we create vectors using `c()` command.
- We can create matrices with `matrix()` command

## Usage

```
matrix(data, nrow, ncol, byrow)
```

- Where:
  - `data`: vector of matrix element
  - `nrow`, `ncol`: number of rows/columns
  - `byrow`: if TRUE, the matrix is filled by rows; if FALSE, it is filled with columns

# Creating Vectors and Matrices: Example

```
# Creating matrices
```

```
A <- matrix(data = c(1, 4, 3, 5), nrow = 2, byrow = TRUE)
B <- matrix(data = c(1, 4, 3, 5), nrow = 2, byrow = FALSE)
C <- matrix(data = c(9, 7, 6, 2, 1, 3), nrow = 2,
            byrow = TRUE)
D <- matrix(data = c(2, 4, 5, 7, 1, 2), nrow = 3,
            byrow = TRUE)
```

```
# Print
```

```
A
```

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

```
B
```

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

## Creating Vectors and Matrices: Example 2

C

```
##      [,1] [,2] [,3]
## [1,]    9    7    6
## [2,]    2    1    3
```

D

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

# Vector Operations

```
# Creating vectors
```

```
a <- c(1, 4, 5, 3, 7)
```

```
b <- c(3, 2, 4, 7, 1)
```

```
c <- c(8, -2, -4)
```

```
# Vector operations
```

```
a + b # vector addition
```

```
## [1] 4 6 9 10 8
```

```
3 * a # scalar product
```

```
## [1] 3 12 15 9 21
```

```
a + 3 # !?
```

```
## [1] 4 7 8 6 10
```



# Vector Operations Continued

```
a %*% b # dot/inner product
```

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

```
## [1,]    59
```

```
a * b # different from above!
```

```
## [1]  3  8 20 21  7
```

- Quick exercise: based on last class's matrix algebra lecture, show how to calculate the dot product of a and b.

# Matrix Operations

```
A - B # matrix addition/subtraction
```

```
##      [,1] [,2]  
## [1,]    0    1  
## [2,]   -1    0
```

```
3 * C # scalar product
```

```
##      [,1] [,2] [,3]  
## [1,]   27   21   18  
## [2,]    6    3    9
```

```
A + 2
```

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

# Matrix Operations Continued

```
A %*% B # matrix product
```

```
##      [,1] [,2]  
## [1,]   17  23  
## [2,]   23  34
```

```
A * B # different from above!
```

```
##      [,1] [,2]  
## [1,]     1  12  
## [2,]    12  25
```

# Matrix Operations 3

```
t(C) %*% B # t() to transpose matrices
```

```
##      [,1] [,2]  
## [1,]   17  37  
## [2,]   11  26  
## [3,]   18  33
```

```
C %*% c # vectors are treated as the k by 1 matrices
```

```
##      [,1]  
## [1,]   34  
## [2,]    2
```

```
t(c) %*% D
```

```
##      [,1] [,2]  
## [1,]    2  10
```

# Determinants and Inverse

```
det(A) # determinant
```

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

```
solve(B) # inverse
```

```
##           [,1]      [,2]  
## [1,] -0.7142857  0.4285714  
## [2,]  0.5714286 -0.1428571
```

- Quick exercise: What happens when we multiply B by solve(b)?

# Solving Systems of Linear Equations

- We can also use the `solve()` command to find solutions to equations.
- Example: Let's solve the following system of equations with R.

$$x + 3y = 7$$

$$2x + 5y = 10$$

```
coefs <- matrix(c(1, 3, 2, 5), nrow = 2, byrow = TRUE)
rhs <- c(7, 10)
solve(coefs, rhs)
```

```
## [1] -5  4
```

# Vector/Matrix Operations: Summary

Command	Meaning
+	Summation
-	Subtraction
*	Element-wise product (Hadamard product)
%*%	Matrix/Vector product
length()	(For vectors) Vector length
dim()	(For matrices) Matrix dimension
t()	Transpose
det()	Determinant
solve()	Inverse
diag(A)	( <b>A</b> is a square matrix) Extract diagonal elements
diag(k)	( <i>k</i> is a scalar) Create a $k \times k$ identity matrix
eigen()	Compute eigenvalues and eigenvectors

# Accessing Vector/Matrix Elements

- For vectors: `vectorname[i]` extracts the *i*th element of the vector
  - We can put in a vector within `[ ]` (e.g `1:3`) to extract multiple elements
  - If we specify negative numbers within `[ ]`, R deletes corresponding elements
- For matrices: `matrixname[i, j]` extracts the element in *i*th row and *j*th column
  - `matrixname[i,]` extracts all the elements in *i*th row as a vector.
  - `matrixname[, j]` extracts all the elements in *j*th column as a vector.



# Accessing Vector/Matrix Elements: Example

```
b[3]
```

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

```
a[c(2, 4)]
```

```
## [1] 4 3
```

```
b[c(-1, -5)]
```

```
## [1] 2 4 7
```

```
C[2, 1]
```

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

```
D[-2,]
```

```
##      [,1] [,2]
```

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

```
## [2,]    1    2
```

# Accessing Vector/Matrix Elements: Example Continued

```
B[1, 1] <- 9
```

```
B
```

```
##      [,1] [,2]  
## [1,]    9    3  
## [2,]    4    5
```

```
C[, 2] <- c(8, 3)
```

```
C
```

```
##      [,1] [,2] [,3]  
## [1,]    9    8    6  
## [2,]    2    3    3
```

# Exercises!

For the following matrix

$$\mathbf{A} = \begin{pmatrix} 7 & -3 & 0 \\ -2 & 6 & 1 \\ 0 & -5 & 6 \end{pmatrix},$$

- 1 Find the determinant
- 2 Calculate the inverse
- 3 Replace the third row to  $(4, 2, 1)$  and recompute the determinant
- 4 Delete the first row and third column and find its inverse matrix