INTRODUCTION TO R

Introduction, basic operators, vectors, matrices

Introduction

What is programming?

What is R?

Universiti Kebangsaan Malaysia
The National University of Malaysia

- R is a programming language and free software environment for statistical computing.

- Supported by the R Foundation for Statistical Computing.
- Widely used among statisticians and data miners for developing statistical software and data analysis.

- Website: https://www.r-project.org/

A bit of history

- First, there was S programming language (1976).
- Then there was S-PLUS (1988), currently owned by TIBCO.
- R is an implementation of the S language.
  - First developed in 1992 at University of Auckland.
  - Named R because the creators Robert Gentleman and Ross Ihaka.
  - Rising in popularity ever since.

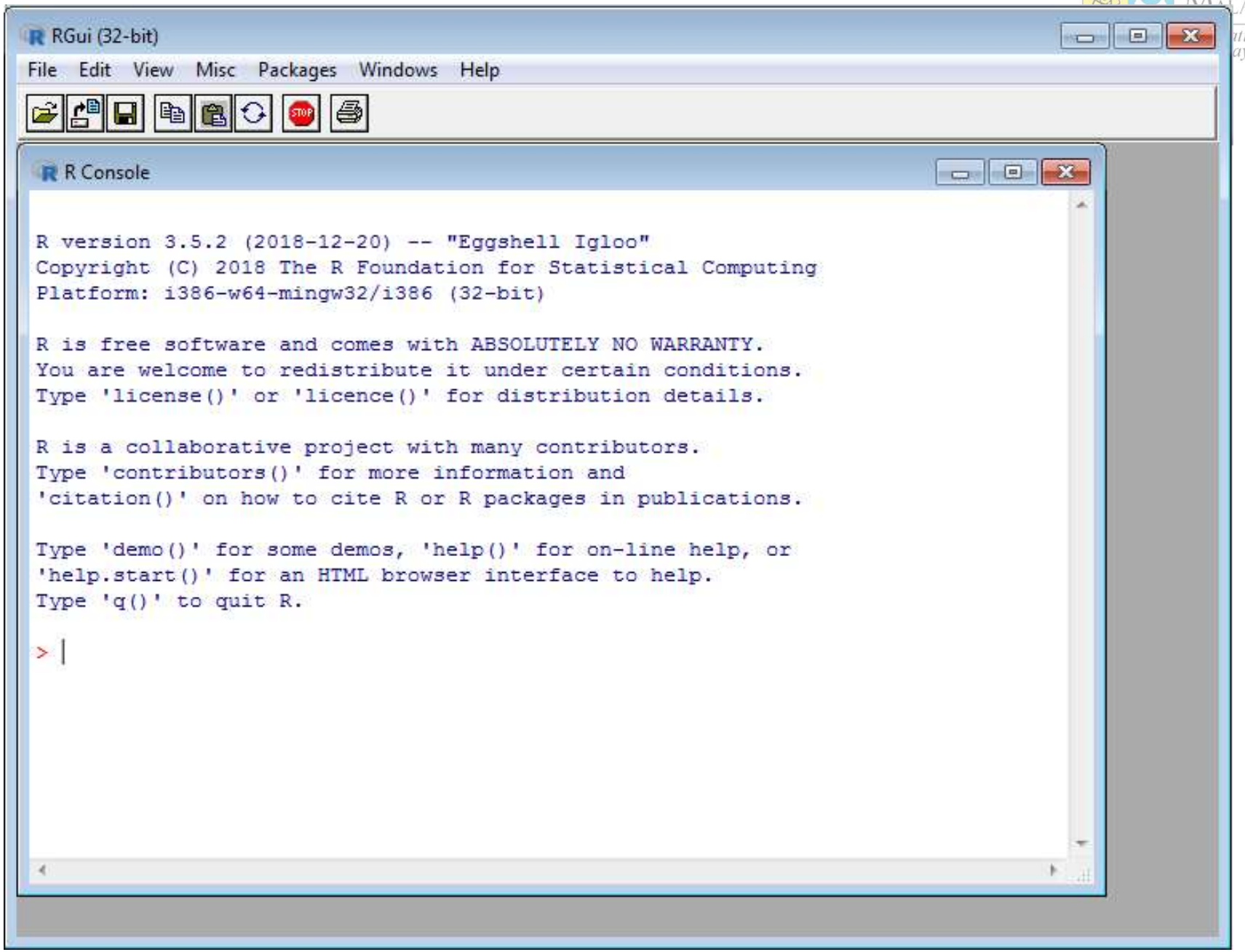
Why R?

- FREE.
- R is getting more popular and used widely in the industry.
- Able to write own code for data analysis.
- Multiple packages (codes written by someone else) available online.

Why R?

- Compared to MS Excel or SPSS:
  - Excel and SPSS are not free.
  - R is less user friendly.
  - But more powerful once you know how to code.
  - Useful especially in more advanced statistics.

- Compare to Python:
  - Python is also free.
  - R is made by statisticians (and mostly for statisticians).
  - Python is made by computer scientists and has more functionality outside of data analysis.
  - R is (mostly) easier for data analysis.



What about RStudio?

- RStudio is an integrated development environment (IDE) for R
- Developed by RStudio, Inc. (now known as Posit, PBC) which has no formal connection to the R Foundation.
- Can be used freely, but there is also paid option for more supports.

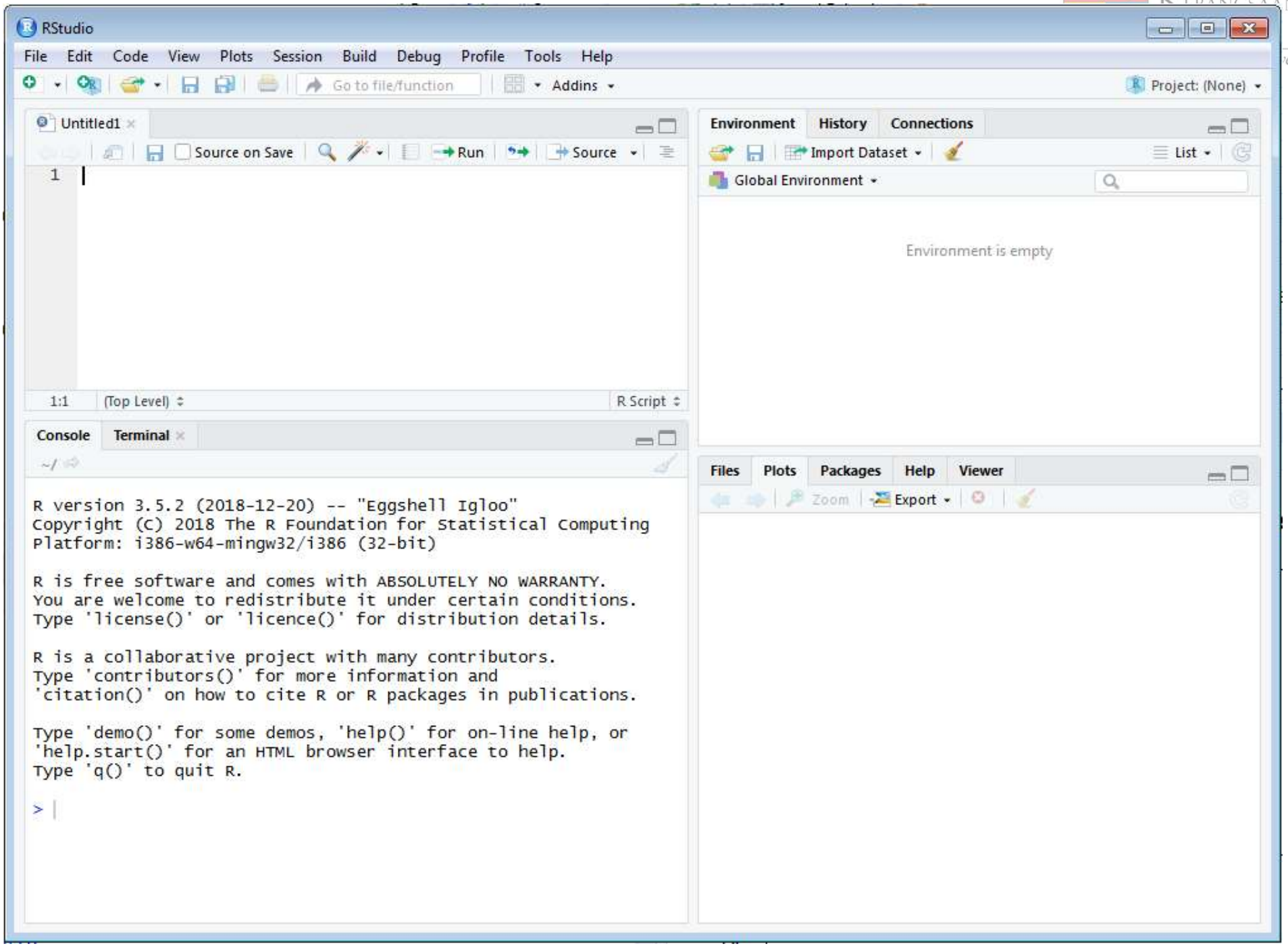
- Website: https://posit.co/

I use RStudio.

- Personal preference.

- Reason:
  - Less clunky.
  - Easier to type in, for example I can use shift+arrow to highlight codes (I can't in R!).
  - Cleaner interface.

- Is RStudio required?
  - No. But I would recommend it.



Installation

- Installing R:
  - Simply go to: https://cran.r-project.org/

- Installing RStudio:
  - Go to: https://posit.co/downloads/
  - Note: RStudio requires installation of R and cannot run independently.

Some basic calculations in R

Basic calculations in R

- R can be used like an advanced calculator.

- Example:

```
> 3+2
[1] 5
> 3*5
[1] 15
> 7/3
[1] 2.333333
```

Basic operations

Operation	Operator/function	Example Input	Example Output
Addition	`+`	`10+2`	`12`
Subtraction	`-`	`9-3`	`6`
Multiplication	`*`	`5*5`	`25`
Division	`/`	`6/3`	`2`
Power	`^`	`5^2`	`25`
Square root	`sqrt()`	`sqrt(4)`	`2`
Exponent, e^x	`exp()`	`exp(1)`	`2.718282`
Log	`log()`	`log(exp(1))`	`1`
Absolute	`abs()`	`abs(-10)`	`10`

Trigonometric operations

Operation	Operator/function	Example Input	Example Output
Sine	`sin()`	`sin(pi)`	`0`
Cosine	`cos()`	`cos(pi)`	`-1`
Tangent	`tan()`	`tan(pi)`	`0`
Inverse sine (arcsin)	`asin()`	`asin(1)`	`1.570796`
Inverse cosine (arccos)	`acos()`	`acos(1)`	`0`
Inverse tangent (arctan)	`atan()`	`atan(1)`	`0.7853982`

- Note:
  - The trigonometric functions use radian, not degree.
  - You can call the value π by using `pi` in your code.

Boolean/logical operators

Operation	Operator/function	Example Input	Example Output
And	`&`	`TRUE & FALSE`	`FALSE`
Or	`|`	`TRUE | FALSE`	`TRUE`
Equals	`==`	`3 == 3`	`TRUE`
Not equals	`!=`	`3 != 3`	`FALSE`
Greater than	`>`	`4 > 3`	`TRUE`
Less than	`<`	`3 < 3`	`FALSE`
Greater than or equals	`>=`	`3 >= 3`	`TRUE`
Less than or equals	`<=`	`4 <= 3`	`FALSE`
Not	`!`	`!(3 > 2)`	`FALSE`

Storing a value to a variable

- Oftentimes, it is easier to assign name/variable to a value.
- Then any calculation can be done using the new assigned variable.
- Note that the upper/lower case in the name of the variable matters.
- To assign a variable to a value, you can use either
  - `variable = value`
  - `variable <- value`
  - `value -> variable`

Storing a value to a variable

- Example:

```
> x = 350
> x
[1] 350
> y <- 100
> y
[1] 100
> 50 -> z
> z
[1] 50
> x + y
[1] 450
> y/z
[1] 2
> sqrt(y)
[1] 10
```

Basic classes of object

- There are a few basic object classes:
  - Numeric: numbers with decimals
  - Integer: integers, it is a subset of numeric
  - Logical: TRUE/FALSE
  - Character: string or text

- You can use `class()` function to check the class of an object.

- You can also change the classes of the object using the following functions:
  - `as.numeric()`
  - `as.integer()`
  - `as.logical()`
  - `as.character()`

Basic classes of object

- Example:

```
> class(3)
[1] "numeric"
> class("3")
[1] "character"
> as.character(3)
[1] "3"
> as.numeric(TRUE)
[1] 1
> as.numeric("A")
[1] NA
Warning message:
NAs introduced by coercion
```

Exercises

1. Calculate these values using R.
   a) $2^2 - 3$
   b) $\sqrt{\sin\left(\frac{\pi}{3}\right)}$
   c) $5\log(4e)$

2. Using R, save the values in part (a), (b) and (c) above as x, y and z, respectively. Then calculate these values:
   a) $\frac{3z}{x}$
   b) $x^y + x^z$

3. Determine whether $3x + 2z$ is greater than or equal to e^y.

Vectors

Working with vectors

- To work with vectors, you will need to use `c()` function. For example, if x is a vector (1,2,3), you need to write down `x <- c(1,2,3)`.

- Then you can use the basic operators as used in previous examples.

- Note that the basic operators are component-wise.

Working with vectors

- Example:

```
> x <- c(1,2,3)
> y <- c(4,5,6)
> x*2
[1] 2 4 6
> x+y
[1] 5 7 9
> x*y
[1]  4 10 18
```

Working with vectors

- Example:

```
> x <- c(1,2,3)
> y <- c(1,3,2)
> x == y
[1]  TRUE FALSE FALSE
> x <= y
[1]  TRUE  TRUE FALSE
> all(x==y)
[1] FALSE
```

- Note: the function `all()` will output TRUE if all the component-wise elements are TRUE, and will output FALSE otherwise.

Selecting elements from a vector

- To select an element out of a vector, we use the `[]` notation.

- For example, if we want to call out the 3rd element of a vector x, we use `x[3]`.

```
> x <- c(3,2,5,6,3)
> x[3]
[1] 5
> x[5]
[1] 3
> x[6]
[1] NA
```

Selecting elements from a vector

- You can also use vectors to select multiple elements of a vector using `x[c(???)]`.

- Example:

```
> x <- c(3,2,5,6,3)
> x[c(3,5,4)]
[1] 5 3 6
```

Selecting elements from a vector

- You can also select elements that fit your desired conditions using `x[conditions]`.

- Example:

```
> x <- c(3,2,5,6,3)
> x>3
[1] FALSE FALSE  TRUE  TRUE FALSE
> x[x>3]
[1] 5 6
> x[x<=3]
[1] 3 2 3
```

Vector operations


```
x <- c(1,5,4)
```

Operation	Operator/function	Example Input	Example Output
Summation	`sum()`	`sum(x)`	10
Product	`prod()`	`prod(x)`	20
Maximum	`max()`	`max(x)`	5
Minimum	`min()`	`min(x)`	1
Length	`length()`	`length(x)`	3
Which of the element is max	`which.max()`	`which.max(x)`	2
Which of the element is min	`which.min()`	`which.min(x)`	1
Sort	`sort()`	`sort(x)`	1 4 5

Some tips and tricks to generating vectors

- In many cases, we want vectors to simply be integers between a and b.
  - To do that, we can use `a:b` to list down all integers between the two numbers.

- Alternatively, using `seq()` function allows us to create a sequence of numbers.
  - `seq(1, 10, by=0.5)` gives a sequence from 1 to 10 where the difference between two consecutive numbers is 0.5.
  - `seq(1, 10, length.out=5)` gives a sequence from 1 to 10 where the length of the sequence is 5.

Some tips and tricks to generating vectors

- Example:

```
> 3:6
[1] 3 4 5 6
> 7:2
[1] 7 6 5 4 3 2
> seq(3, 5, by=0.3)
[1] 3.0 3.3 3.6 3.9 4.2 4.5 4.8
> seq(3, 4, length.out=6)
[1] 3.0 3.2 3.4 3.6 3.8 4.0
> seq(3, 4, length.out=5)
[1] 3.00 3.25 3.50 3.75 4.00
```

Other tips and tricks for generating vectors

- Another useful function is `rep()`.
  - It produces a replicate of values.
  - `rep(0,10)` gives a vector of 0's with length 10

- Also, there is a function called `vector()` which produces vector of given mode/class and length
  - `vector("numeric",10)` gives vector of 0's with length 10
  - `vector("logical",10)` gives vector of FALSE with length 10

Other tips and tricks for generating vectors

- Example

```
> rep(0,10)
 [1] 0 0 0 0 0 0 0 0 0 0
> rep(c(1,2), 5)
 [1] 1 2 1 2 1 2 1 2 1 2
> rep(c(1,2), each=5)
 [1] 1 1 1 1 1 2 2 2 2 2
> vector("numeric",5)
 [1] 0 0 0 0 0
> vector("character",7)
 [1] "" "" "" "" "" "" ""
```

Exercises

1. Construct a vector consisting of a sequence of numbers from 2 to 10 where the length is 30.
   a) What is the sum of the vector generated?
   b) What is the 15^{th} element of the vector?
   c) How many of the elements in the vector is greater than 5?

Matrices

Working with matrices

- To create a matrix, the function `matrix()` must be used.

- This function requires the elements of the matrix written in vector format, and requires we set the number of columns/rows.

Working with matrices

- Example:

```
> x <- c(1,2,3,4,5,6)
> X <- matrix(x, nrow=3)
> X
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> X <- matrix(x, nrow=3, byrow=TRUE)
> X
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
```

Selecting elements of a matrix

- Selecting elements of a matrix uses the `[]` notation, like what we used in vectors.
  - `X[i,j]`: gives the element in the ith row and jth column
  - `X[,j]`: gives the elements in the jth column
  - `X[i,]`: gives the elements in the ith row

Selecting elements of a matrix

- Example:

```
> X <- matrix(c(1,2,3,4,5,6), nrow=3,
      byrow=TRUE)
> X
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
> X[3,1]
[1] 5
> X[,1]
[1] 1 3 5
> X[2,]
[1] 3 4
```

Matrix operations

Operation	Operator/function	Example Input
Matrix multiplication	`%*%`	`A %*% B`
Inverse of a matrix	`solve()`	`solve(A)`
Transpose	`t()`	`t(A)`
Number of rows	`nrow()`	`nrow(A)`
Number of columns	`ncol()`	`ncol(A)`
Row bind	`rbind()`	`rbind(A,B)`
Column bind	`cbind()`	`cbind(A,B)`

- Note: using the * operator on matrix will result in component-wise multiplication.

Matrix operations

- Example:

```
> A <- matrix(c(1,2,3,4), nrow=2, byrow=TRUE)
> B <- matrix(c(1,0), ncol=1)
> A
     [,1] [,2]
[1,]    1    2
[2,]    3    4
> B
     [,1]
[1,]    1
[2,]    0
> A %*% B
     [,1]
[1,]    1
[2,]    3
```

Matrix operations

```
> solve(A)
     [,1] [,2]
[1,] -2.0  1.0
[2,]  1.5 -0.5
> A*A
     [,1] [,2]
[1,]    1    4
[2,]    9   16
> A %*% A
     [,1] [,2]
[1,]    7   10
[2,]   15   22
> t(A)
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

```
> A
     [,1] [,2]
[1,]    1    2
[2,]    3    4
> B
     [,1]
[1,]    1
[2,]    0
```

Matrix operations


```
> cbind(A,solve(A))
     [,1] [,2] [,3] [,4]
[1,]    1    2 -2.0  1.0
[2,]    3    4  1.5 -0.5
> rbind(A,solve(A))
     [,1] [,2]
[1,]  1.0  2.0
[2,]  3.0  4.0
[3,] -2.0  1.0
[4,]  1.5 -0.5
```

```
> A
     [,1] [,2]
[1,]    1    2
[2,]    3    4
> solve(A)
     [,1] [,2]
[1,] -2.0  1.0
[2,]  1.5 -0.5
```

Modifying an element of a vector/matrix

- You can modify elements of a vector/matrix individually using the assign notation.

```
> x <- c(1,2,3,4)
> x[3] <- 10
> x
[1]  1  2 10  4
> X <- matrix(x, nrow=2, byrow=TRUE)
> X
     [,1] [,2]
[1,]    1    2
[2,]   10    4
> X[2,] <- c(0,5)
> X
     [,1] [,2]
[1,]    1    2
[2,]    0    5
```

Exercises

1. Let

$$X = \begin{bmatrix} 1 & 5 & 1 \\ 10 & 4 & 1 \end{bmatrix}$$

   a) Assign the variable `X` as the matrix above.
   b) Using R, extract the element in the 2nd row and 1st column.
   c) Using R, extract the whole 2nd column.

2. Let

$$Y = \begin{bmatrix} 3 & 15 \\ 5 & 2 \end{bmatrix}$$

   a) Assign the variable `Y` as the matrix above.
   b) Calculate Y^{-1}.
   c) Calculate $X^T Y$.

What if things go wrong?

- To read documentation of a function, you can use the `help()` function or `?` function.
- Eg:
  - `help(sum)`
  - `?sum`

- If everything fails, Google. Yes, Google.

Functions & descriptions

Function	Description
`sqrt(), exp(), log()`	Basic operations
`class()`	Check class of an object
`as.numeric(), as.character()`	Change the class of an object
`c()`	Create a vector
`max(), min(), sum(), length(), sort(), which.max(), which.min()`	Vector operations
`seq()`	Create a sequence from a number to another number
`rep()`	Create a new vector by repeating a number or vector
`matrix()`	Create a matrix
`solve(), t(), rbind(), cbind()`	Matrix operations
`help()`	Find functions/data documentation