Lecture 1.6 – Vectors in R

Specific Learning Objectives:

1.1.9 – Create vectors, arrays, matrices, lists, and data frames.

1.1.10 – Understand vectors and vectorized calculations.

1.1.11 – Understand the data classes of R.

1.1.12 – Learn how to index vectors, arrays, matrices, lists, and data frames.

Question

- In physics, what is the difference between a vector and a scalar?

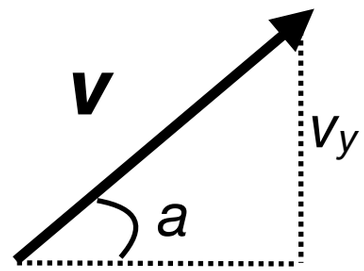
- Values in physics:

<u>scalar:</u>

v = 3

- magnitude
- no direction

<u>vector:</u>



magnitude = 3

$\boldsymbol{v} = \langle v_x, v_y \rangle$

$\boldsymbol{v} = \langle 3 \cos a, 3 \sin a \rangle$

- magnitude
- direction

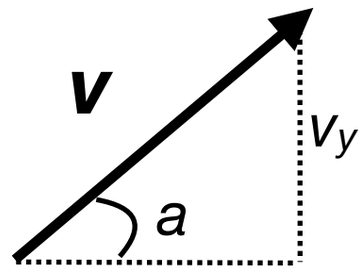
Question

- How is this representation different in R?

- Values in physics:

scalar: $v = 3$

vector: $\boldsymbol{v} = \langle v_x, v_y \rangle$



- Vectors in R:

scalar: `> v.scalar <- 3`

vector: `> v.vector <- c(1,2)`

Values		
vector length 1 →	v.scalar	3
vector length 2 →	v.vector	num [1:2] 1 2

All numbers are vectors in R!

Properties of Vectors in *R*

- Vectors in *R* have basic properties:

  - Vectors are a special type of **array** made of *integer*, *numeric,* or *logical* (TRUE/FALSE) elements.

Vector:

integers

$$\vec{v} = \begin{bmatrix} 1 & 7 & 9 \end{bmatrix}$$

Not a vector: array with characters

$$\vec{v} = \begin{bmatrix} 1 & \text{flower} & 9 \end{bmatrix}$$

character string

  - Vectors can be 1 or more elements in length, but must be 1-dimensional. (Matrices are *2*-dimensional.)

Vector:

1 dimension

$$\vec{v} = \begin{bmatrix} 1 & 7 & 9 \end{bmatrix}$$

Not a vector: 2-dimensional matrix

1 dimension

$$\mathbf{m} = \begin{bmatrix} 1 & 7 & 9 \\ 2 & 6 & 3 \\ 8 & 1 & 4 \end{bmatrix}$$

1 dimension

Making Vectors in R

- Concatenate function `c()`:

```
> v.vector <- c(1,2)
```

  - Arguments are objects, to be bound together to create one vector

  - Unlimited number of arguments

What happens if you try to make the following vector with `c()`?

$$\vec{v} = \begin{bmatrix} 1 & \text{flower} & 9 \end{bmatrix}$$

```
v <- c(1, "flower", 9)
```

R doesn't like to mix data types in arrays!

Making Vectors in R

- Sequences:

  - Simple sequences with **`seq()`**

```
> vec <- seq(1,10)
```

from **to**

=

  - Simple sequences with :

```
> vec2 <- 1:10
```

from:to

  - Sequence with specific step size with **`seq()`**

```
> vec3 <- seq(1, 10, 0.5)
```

from **to** **by**

  - Sequence with specific output length with **`seq()`**

```
> vec4 <- seq(1, 10, length.out = 50)
```

from **to**

total length of output vector

Making Vectors in R

- Repetitions with **`rep()`**

  - Repeat a number into a longer vector

```
> repvec <- rep(1, 10)
```

number to copy — **times to copy it**

  - Repeat an object into a longer vector

```
> vec1 <- c(1,8,10,3)

> repvec2 <- rep(vec1, times=4)
```

vector to copy — **times to copy it**

- Repeat an object into a longer vector, but do it by element

```
> repvec3 <- rep(vec1, times=3, each=2)
```

```
> repvec2
 [1]  1  8 10  3  1  8 10  3  1  8 10  3  1  8 10  3
> repvec3
 [1]  1  1  8  8 10 10  3  3  1  1  8  8 10 10  3  3  1  1  8  8 10 10  3  3
```

Check Your Understanding

Write one line of code to recreate this vector and store it as the object **`boop`**

```
 [1] 1 1 1 2 2 2 3 3 3 4 4 4 5 5 5 1 1 1 2 2 2 3 3 3 4 4 4 5 5 5 1 1 1 2 2 2 3 3 3 4 4 4 5
[44] 5 5
```

Write a different line of code that reproduces this vector and store it as the object **`boop2`**

Test whether or not **`boop`** and **`boop2`** are the same using:

```
all.equal(boop, boop2)
```

Working with Vectors

- Vectors can be used for many things! Sometimes it's useful to find out information about them.

  - Find length of a vector with **`length()`**

```
> length(boop)
[1] 45
```

  - Find what type of data the vector has with **`class()`**

```
> class(boop)
[1] "numeric"
```

Why is `boop` numeric instead of integer?

Indexing Vectors

- **Indexing**: reference a specific element of a vector

```
> blop <- seq(2,20,by=2)
> blop
 [1]  2  4  6  8 10 12 14 16 18 20
```

element number: 1 2 3 4 5 6 7 8 9 10

Square brackets are used to index vectors!

  - One element

```
> blop[1]
[1] 2
```

```
> blop[4]
[1] 8
```

  - Sequence of elements

```
> blop[6:9]
[1] 12 14 16 18
```

  - Multiple elements

```
> blop[c(5,7,10)]
[1] 10 14 20
```

  - All but one element

```
> blop[-3]
 [1]  2  4  8 10 12 14 16 18 20
```

Vectorized Calculations

- Vectors have special properties for most basic math operations, called *vectorized calculations*.

  - Calculations in *R* on vectors take place element-wise:

```
> blop
 [1]  2  4  6  8 10 12 14 16 18 20
      *  *  *  *  *  *  *  *  *  *
      3  3  3  3  3  3  3  3  3  3
      =  =  =  =  =  =  =  =  =  =
> blop*3
 [1]  6 12 18 24 30 36 42 48 54 60

> blop+3
 [1]  5  7  9 11 13 15 17 19 21 23
```

Why is this useful?
Because otherwise you'd have to do each of these calculations one-by-one, which is slow!
Vectorized functions are **very fast**!

Vectorized Calculations

- When elements are the same length:

```
> blop
 [1]  2  4  6  8 10 12 14 16 18 20
```

```
> blop - c(0.4, 0.8, 1.2, 0.2, 0.75, 1.0, 1.0, 2.0, 0.45, 0.5)
```

```
 2   4   6   8 10 12 14 16 18 20
 -   -   -
0.4 0.8 1.2
 =   =   =
1.60 ?   ?
```

```
> blop - c(0.4, 0.8, 1.2, 0.2, 0.75, 1.0, 1.0, 2.0, 0.45, 0.5)
 [1]  1.60  3.20  4.80  7.80  9.25 11.00 13.00 14.00 17.55 19.50
```

Vectorized Calculations

- When elements are different lengths:

```
> blop
 [1]  2  4  6  8 10 12 14 16 18 20

> blop-c(1,2)
```

```
 2  4  6  8 10 12 14 16 18 20
 -  -  -  -
 1  2  1  2
 =  =  =  =
 1  2  5  6
```

```
> blop-c(1,2)
 [1]  1  2  5  6  9 10 13 14 17 18
```

Special Functions on Vectors

How do you know which functions? Take a look at the help doc!

- Some functions are special to vectors:
  - Sum the elements of a vector with **`sum()`**

```
> sum(blop)
[1] 110      =  2 + 4 + 6 + 8 +10+12+14+16+18+20
```

  - Multiply the elements of a vector with **`prod()`**

```
> prod(blop)
[1] 3715891200  =  2 x 4 x 6 x 8x10x12x14x16x18x20
```

  - Calculate the mean of elements of a vector with **`mean()`**

```
> mean(blop)
[1] 11
```

  - Calculate the standard deviation of elements of a vector with **`sd()`**

```
> sd(blop)
[1] 6.055301
```

Check Your Understanding

1. The vector `blippi` has 12 elements:

```
> blippi
 [1]  2  8  9 19 39  1  9  3 48 10 23 87
```

What is the correct way to positions 3, 7, and 12 in a single line of code?

a) `blippi(3,7,12)`

b) `blippi(c(3,7,12))`

c) `blippi[3,7,12]`

d) `blippi[c(3,7,12)]`

Correct answer

2. Why don't a, b, or c work? Explain why!

3. Write the code to multiply each position in `blippi` by 10.

Action Items

1. **Complete assignment 1.7.**

2. **Read Davies Ch. 3 for next time.**