Vectors and Matrices in R

Matteo Fasiolo

Creating Vectors in R

Using the "combine" operator c:

```r
vd <- c(1,2,3,4)
vd
```

```
[1] 1 2 3 4
```

```r
vi <- c(1L, 2L, 3L, 4L)
vi
```

```
[1] 1 2 3 4
```

This is not quite the same:

```r
class(vd)
```

```
[1] "numeric"
```

```r
class(vi)
```

```
[1] "integer"
```

Creating vectors of a given size and type:

```r
vd <- numeric(4)
vd
```

```
[1] 0 0 0 0
```

```r
vi <- integer(4)
vi
```

```
[1] 0 0 0 0
```

```r
vl <- logical(4)
vl
```

```
[1] FALSE FALSE FALSE FALSE
```

```r
vc <- character(4)
vc
```

```
[1] "" "" "" ""
```

The : symbol generates a vector of a range of values:

```
a <- 1:3
a
```

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

We can also reverse

```
3:-3
```

```
[1]  3  2  1  0 -1 -2 -3
```

or use variables to define the range:

```
a <- 0
b <- 3
a:b
```

```
[1] 0 1 2 3
```

Ambiguous cases:

```
1.1:3.1
```

```
[1] 1.1 2.1 3.1
```

```
1.1:3
```

```
[1] 1.1 2.1
```

Often better to use seq:

```
seq(1.5, 3.5, by = 0.5)
```

```
[1] 1.5 2.0 2.5 3.0 3.5
```

```
seq(3.5, 1.5, by = -0.5)
```

```
[1] 3.5 3.0 2.5 2.0 1.5
```

```
seq(1.5, 3.5, length.out = 6)
```

```
[1] 1.5 1.9 2.3 2.7 3.1 3.5
```

Vector Operations

You can find reference to most of these from ART 2.4

Vector addition adds each dimension of vectors.

```
a <- c(1,2,3,4)
b <- c(1,2,3,4)
( c <- a + b )  # Same as c <- a + b; c
```

```
[1] 2 4 6 8
```

Vector Subtraction −, Multiplication ∗, Division / works the same way.

Do not confuse ∗ with %∗%, ∗ is dimension-wise multiplication.

Logical operations performed on each element of the vector:

```r
a <- c(1,2,3,4)
a > 2
```

```
[1] FALSE FALSE  TRUE  TRUE
```

Many math functions operate on each element of the input vector (`sin,cos,exp,log`):

```r
exp(c(1,2,3,4))
```

```
[1]  2.718282  7.389056 20.085537 54.598150
```

This type of vector operations are called **vector-to-vector**.

An example of a **vector-to-scalar** computation is:

```r
a <- c(1,2,3,4)
sum(a)
```

```
[1] 10
```

and an example of **vector-to-matrix** computation is:

```r
b <- c(4,3,2,1)
tcrossprod(a, b)
```

```
     [,1] [,2] [,3] [,4]
[1,]    4    3    2    1
[2,]    8    6    4    2
[3,]   12    9    6    3
[4,]   16   12    8    4
```

which is computing $\mathbf{a}\mathbf{b}^T$.

Indexing Element(s) in a Vector

Use [] to index a single element.

Indexing start from 1 not 0!

Indexing element that beyond the range of the vector returns NA.

```
a <- c(1,2,3,4)
a[5]
```

```
[1] NA
```

and doing

```
a[5] <- 10
a
```

```
[1]  1  2  3  4 10
```

works! Quite different from C.

Generally better to allocate memory in advance:

```r
tic <- Sys.time()
x <- c(1)
for(ii in 2:1e6){
  x[ii] <- ii
}
Sys.time() - tic
```

```
Time difference of 0.2227435 secs
```

```r
tic <- Sys.time()
x <- numeric(1e6)
for(ii in 2:1e6){
  x[ii] <- ii
}
Sys.time() - tic
```

```
Time difference of 0.05552578 secs
```

You can index more than one elements at a time:

```r
a <- c("a", "b", "c", "d")
b <- c(1,3,4)
a[b]
```

```
[1] "a" "c" "d"
```

Note: strictly speaking `b` is a vector of doubles!

Let's try:

```r
a[c(1.6, 3.6)]
```

```
[1] "a" "c"
```

Potentially leading to unexpected results. This is because:

```r
as.integer(c(1.6, 3.6))
```

```
[1] 1 3
```

See also `as.double`, `as.logical`, `as.character`, …

Use : to access multiple contiguous elements in a vector.

```
a[1:3] # same as a[c(1, 2, 3)]
```

```
[1] "a" "b" "c"
```

You can also use conditional expression to index a vector:

```
a <- c(7,2,1,5)
a[a>2]
```

```
[1] 7 5
```

Break it down step by step:

```
(l <- a > 2)
```

```
[1]  TRUE FALSE FALSE  TRUE
```

```
a[l]
```

```
[1] 7 5
```

Note: **common source of errors**:

```r
l
```

```
[1]  TRUE FALSE FALSE  TRUE
```

```r
class(l)
```

```
[1] "logical"
```

Suppose that at some point in your code:

```r
l <- as.integer(l)
l
```

```
[1] 1 0 0 1
```

Then this might not be what you expected:

```r
a[l]
```

```
[1] 7 7
```

You are selecting `a[1]` twice, not `a[c(1, 4)]`.

Exploring R's documentation

How to get documentation on `*`, `+`, `:` etc?

For functions we do:

```
?mean
```

This won't work:

```
?*
# Error: unexpected '*' in "?*"
```

We need to type

```
?"*"
?":"
?"?"
?"%*%"
```

Matrix Construction

You can create a matrix from a vector using `matrix` function:

```r
a <- c(1,2,3,4)
A <- matrix(a, nrow = 2)
A
```

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

`nrow` parameter specifies the number of rows in the matrix.

```r
a <- c(1,2,3,4)
A <- matrix(a, nrow = 1)
A
```

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

By default, `matrix` treats your input vector `1:4` as a **column-major order**.

If you want to fill the matrix in **rows-major order**:

```
A <- matrix(1:4, nrow = 2, byrow = T)
A
```

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

You can get dimension of a matrix using `dim` function.

```
dim(A)
```

```
[1] 2 2
```

Otherwise you can use `ncol(A)` and `nrow(A)`.

Indexing Elements in a Matrix

```
( A <- matrix(1:4, nrow = 2) )
```

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

```
A[1,1]
```

```
[1] 1
```

```
A[1:2,1] # Not a matrix anymore!
```

```
[1] 1 2
```

You can access i-th row of a matrix by using `[i,]`.

```
A[1,] # Do A[ , 1] for first column
```

```
[1] 1 3
```

As for vectors you can index using logical vectors:

```r
( A <- matrix(1:9, nrow = 3) )
```

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

```r
lr <- c(TRUE, FALSE, TRUE)
A[lr, ]
```

```
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    3    6    9
```

```r
lc <- c(TRUE, FALSE, TRUE)
A[lr, lc]
```

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

To access the diagonal do:

```
( A <- matrix(1:9, nrow = 3) )
```

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

```
diag(A)
```

```
[1] 1 5 9
```

To over-write it:

```
diag(A) <- 0
A
```

```
     [,1] [,2] [,3]
[1,]    0    4    7
[2,]    2    0    8
[3,]    3    6    0
```

Matrix Operators

The `+`, `-`, `*`, `/` symbols work in an element-wise fashion.

```r
A <- matrix(1:4, nrow = 2, byrow = T)
B <- matrix(1:4, nrow = 2, byrow = T)
print(A+B)
```

```
     [,1] [,2]
[1,]    2    4
[2,]    6    8
```

The %*% symbol is used for matrix multiplication

```r
print(A%*%B)
```

```
     [,1] [,2]
[1,]    7   10
[2,]   15   22
```

Some subtleties

An R vector is not a column vector (i.e., a matrix with 1 column):

```
A <- matrix(1:4, nrow = 2, byrow = T)
B <- matrix(1:2, nrow = 2, byrow = T)
( v <- A %*% B ) # NOT the same as c(5, 11)
```

```
     [,1]
[1,]    5
[2,]   11
```

```
dim(v)
```

```
[1] 2 1
```

```
( v <- as.vector(v) ) # Same as c(5, 11)
```

```
[1]  5 11
```

```
dim(v) # instead, use length(v) to get number of elements
```

```
NULL
```

R "recycles" without telling you:

```r
a <- c(1, 1, 1, 1)
b <- c(1, 2)
a * b # Same as a * c(b, b)
```

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

This issues a warning:

```r
b2 <- c(1, 2, 3)
a * b2 # Same as a * c(b2, b2[1])
```

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

```
# Warning: longer object length is not a multiple of
# shorter object length
```

Matrix Transposition

Matrix transpose is done by `t` function:

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

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

```r
t(a)
```

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

Note this behaviour:

```r
b <- c(1, 2, 3)
b
```

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

```r
t(b)
```

```
     [,1] [,2] [,3]
[1,]    1    2    3
```

```r
t(t(b))
```

```
     [,1]
[1,]    1
[2,]    2
[3,]    3
```

Again, in R a vector constructed with `c()`, `a:b`, `numeric(n)` is not the same as a matrix with one column. **Big source of errors**!

Matrix Inversion

Recall given a square, full-rank matrix $\mathbf{A}$, $\mathbf{A}^{-1}$ is such that

$$\mathbf{A}^{-1}\mathbf{A} = \mathbf{I}.$$

In R we use the `solve` function:

```r
A <- matrix(1:4, nrow = 2)
solve(A)
```

```
     [,1] [,2]
[1,]   -2  1.5
[2,]    1 -0.5
```

```r
A %*% solve(A)
```

```
     [,1] [,2]
[1,]    1    0
[2,]    0    1
```

You should never invert a matrix unless you have to $(\mathbf{Ax} = \mathbf{b})$!

Conclusion

To create vectors you can use:

- `c(x1, x2, ...)`
- `numeric(n)`, `integer(n)`, ...
- `a:b`
- `seq(a, b, by = eps)` or `seq(a, b, length.out = n)`

Vector and matrix indexing (or subsetting) via integer or logical vectors.

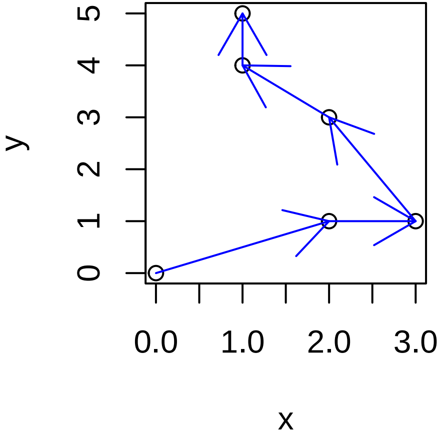
Distinction between elements-wise and vector or matrix operations.

Homework

Problem: given a sequence of locations in space:

a. Find the length and direction of the segments separating each pair of locations.

b. Find the total length of the path.



Homework: part 1

Start by writing a function `dist(a, b)`, where `a` and `b` are vectors represent two points in a space.

`dist` outputs the Euclidean distance between two points.

Requirements:

- Your function should work for input vectors in **any dimension**.
- `dist(v,0)` will return the length of vector `v`.
- **No for-loops**.
- Make test cases, and check your function.

Homework: part 2

Write an R function `angle`, takes two vectors: `a` and `b`.

It outputs the angle between `a` and `b` in degrees.

- The result should be returned **in degrees**, not in radian.
- Use `dist` function you just wrote.
- Make test cases, and check your function.
- Hint: Type `?acos` in command line and read the documentation.

Homework: part 3 (submit)

Consider a sequence of n location in d-dimensional space. E.g.:

```
     x y
loc1 0 0
loc2 2 1
loc3 3 1
loc4 2 3
loc5 1 4
loc6 1 5
```

Write a function that, given a matrix of locations in 2D, returns:

1. a vector containing the length (Euclidean norm) of each step
2. a vector containing the orientation of each step
   - for orientation you should use the angle between the x-axis and the step
3. a scalar representing the total length of the path

Hint: to write a function that returns several objects you can use:

```
my_function <- function(){
  a <- 1:2
  b <- "Hello!"
  return( list(a, b) )
}

x <- my_function()
x[[1]]
```

```
[1] 1 2
```

```
x[[2]]
```

```
[1] "Hello!"
```

That's all you need to know about lists for the moment!

Hint: if you want to check your code visually you can do:

```
locs <- matrix(c(0, 2, 3, 2, 1, 1,
                 0, 1, 1, 3, 4, 5), 6, 2)

plot(locs)

# YOUR CODE
for(ii in list_of_segments){
  # YOUR CODE to compute start (x0, y0) and
  # end (x1, y1) of segments
  arrows(x0, y0, x1, y1)
}
```

Challenge (do not submit): Can you extend your function beyond 2D? Can you do this in C++?

Hint: in D $>$ 2 you need more than one angle to determine the direction of a vector!