Introduction to R

Hui Jiang

jianghui@umich.edu

Data Types

- Logical
- Integer (Factor)
- Double
- Character
- …

Working with Data Types

```
> 1+1
[1] 2
> "1"==1
[1] TRUE
> "1"+1
Error in "1" + 1 : non-numeric argument to binary operator
> T==1
[1] TRUE
> T=="1"
[1] FALSE
> as.numeric(T)=="1"
[1] TRUE
```

Data Structures

Dimension	Homogeneous	Heterogeneous
1	Vector	List
2	Matrix	Data Frame
N	Array	

Related functions: typeof, class, str, dput

Working with Vectors

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

Working with Vectors (cont.)

```
> x=c(3,2,1,2,4)
> sort(x)
[1] 1 2 2 3 4
> sort(x,decreasing=T)
[1] 4 3 2 2 1
> rev(x)
[1] 4 2 1 2 3
> unique(x)
[1] 3 2 1 4
> table(x)
x
1 2 3 4
1 2 1 1
```

Working with Vectors (cont.)

```
> c(2,4,6)
[1] 2 4 6
> 2:6
[1] 2 3 4 5 6
> seq(2,4,by=0.5)
[1] 2.0 2.5 3.0 3.5 4.0
> rep(1:3,times=2)
[1] 1 2 3 1 2 3
> rep(1:3,each=2)
[1] 1 1 2 2 3 3
```

Working with Vectors (cont.)

```
> x=1:6
> x*2
[1]  2  4  6  8 10 12
> x+1
[1] 2 3 4 5 6 7
> x+1:6
[1]  2  4  6  8 10 12
> x*1:6
[1]  1  4  9 16 25 36
> x*1:2
[1]  1  4  3  8  5 12
> x*1:4
[1]  1  4  9 16  5 12
Warning message:
In x * 1:4 :
  longer object length is not a multiple of shorter object length
```

Working with Matrices

```
> m=matrix(1:4, nrow=2, ncol=2)
> m
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> m[1,]
[1] 1 3
> m[1,,drop=F]
     [,1] [,2]
[1,]    1    3
> m[1,1]
[1] 1
```

Working with Matrices (cont.)

```
> m*m
     [,1] [,2]
[1,]    1    9
[2,]    4   16
> m%*%m
     [,1] [,2]
[1,]    7   15
[2,]   10   22
> t(m)
     [,1] [,2]
[1,]    1    2
[2,]    3    4
```

Working with Matrices (cont.)

```
> nrow(m)
[1] 2
> ncol(m)
[1] 2
> dim(m)
[1] 2 2
> cbind(m,c(5,6))
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> rbind(m,c(5,6))
     [,1] [,2]
[1,]    1    3
[2,]    2    4
[3,]    5    6
```

Working with Lists

```
> l=list(x=1:5, y=c('a','b'))
> l[[2]]
[1] "a" "b"
> l[1]
$x
[1] 1 2 3 4 5
> l$x
[1] 1 2 3 4 5
> l['y']
$y
[1] "a" "b“
> length(l)
[1] 2
```

Working with Data Frames

- Data frames are lists of vectors of the same length

```
> x=data.frame(x=1:3, y=c('a','b','c'))
> x
  x y
1 1 a
2 2 b
3 3 c
> x[1:2,1]
[1] 1 2
> x[1:2,1,drop=F]
  x
1 1
2 2
```

Loops

```
> for (i in 1:4) {print(i^2)}
[1] 1
[1] 4
[1] 9
[1] 16
> i=1
> while(i<5){
+ print(i)
+ i=i+1
+ }
[1] 1
[1] 2
[1] 3
[1] 4
```

Conditions

```
> i=5
> if(i>3) {
+ print('Yes')
+ }else{
+ print('No')
+ }
[1] "Yes"
```

Functions

```
> sq <- function(x) {
+ return (x^2)
+ }
> sq(2)
[1] 4
> sq(1:5)
[1]  1  4  9 16 25
```

Vectorization

```
> x=matrix(rnorm(1000000*10), 1000000, 10)
> dim(x)
[1] 1000000      10
> y=rep(0,10)
> for (i in 1:10) {
+ for (j in 1:1000000) {
+ y[i] = y[i] + x[j,i] } }
> y
 [1]  -173.4330  -746.8764   699.4386   777.7185   -71.1766
 [6]   430.3212   295.3495  1485.3183 -2505.8576   574.7012
> apply(x,2,sum)
 [1]  -173.4330  -746.8764   699.4386   777.7185   -71.1766
 [6]   430.3212   295.3495  1485.3183 -2505.8576   574.7012
> colSums(x)
 [1]  -173.4330  -746.8764   699.4386   777.7185   -71.1766
 [6]   430.3212   295.3495  1485.3183 -2505.8576   574.7012
```

Vectorization (cont.)

```
> system.time(for (i in 1:10) {
+ for (j in 1:1000000) {
+ y[i] = y[i] + x[j,i] } })
   user  system elapsed
  15.89    0.00   15.95
> system.time(apply(x,2,sum))
   user  system elapsed
   0.17    0.01    0.18
> system.time(colSums(x))
   user  system elapsed
      0       0       0
```

Pre-allocate Memory

```
> x=NULL
> system.time(for(i in 1:100000) {
+ x=c(x,i)})
 user  system elapsed
  9.48    0.00    9.60
> x=rep(0,100000)
> system.time(for(i in 1:100000){
+ x[i]=i})
 user  system elapsed
  0.10    0.00    0.09
> system.time(x<-1:100000)
   user  system elapsed
      0       0       0
```

References

- Cheatsheets: https://www.rstudio.com/resources/cheatsheets/
- Slides by Phil Boonstra

Exercise

- Write an R function CumSum(x), which computes the cumulative sum of a vector x:

```
CumSum <- function(x) {
    …
    return (y)
}
```

where the output y is a vector that has the same length as x, and for each i, y[i] = x[1]+x[2]+…+x[i].

- Compare the speed of your function with R function cumsum, with input vector of different sizes.