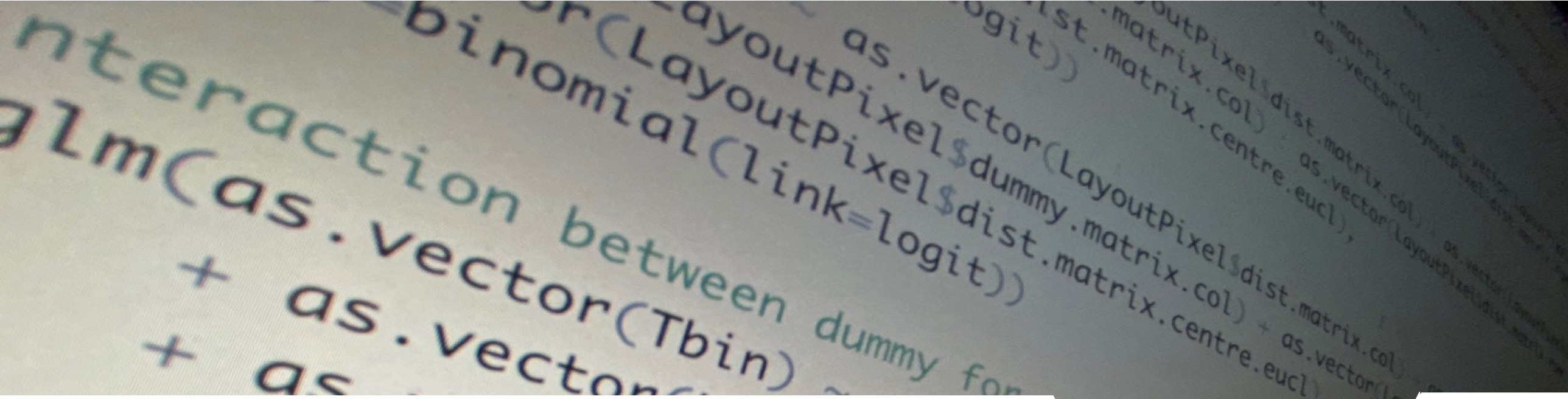

ST 117

2. Basic R

WARWICK

Lecture 4
(Week 2)

Random variables and sampling
Vectors
Lists
Arrays & matrices
Functions

Random variables and sampling

R has excellent built-in routines for simulating from a wide range of distributions.

Binomial

```
> rbinom(10,8,.5)
 [1] 5 5 5 3 7 4 6 3 2 4
> rbinom(10,8,.5)
 [1] 4 3 4 7 3 2 4 7 5 2
```

Poisson

```
> rpois(10,2.7)
 [1] 6 2 3 1 7 1 4 1 3 2
```

Geometric

```
> rgeom(10,0.2)
 [1] 0 0 0 1 1 7 0 0 4 4
> rgeom(10,0.8)
 [1] 0 0 0 0 2 0 0 0 0 3
```

Normal

```
> rnorm(6)
[1] -0.201672239 -0.938337516  1.241665644
  -0.221508994  0.542453444  0.008129779
> rnorm(6,2,4)
[1]  7.209486 -1.919023 -1.863232
   5.379270   2.784150  2.285561
```

Uniform

```
> runif(5)
[1] 0.4200739 0.2142666 0.2348059
   0.2760098 0.6303858
```

Exponential

```
> rexp(5,3)
[1] 0.2199351 0.3435677 0.4999797
   0.3624121 0.8394378
```

Functions for calculating densities, cdfs, and quantiles - show below for normal (continuous) and Poisson (discrete)

cdfs

```
> pnorm(0)
[1] 0.5
> pnorm(-2)
[1] 0.02275013
> pnorm(-4,0,2)
[1] 0.02275013
```

```
> ppois(5,10)
[1] 0.06708596
> ppois(10,5)
[1] 0.9863047
> ppois(5,2.7)
[1] 0.9432683
> ppois(2.7,5)
[1] 0.124652
> ppois(2,5)
[1] 0.124652
```

densities

```
> dnorm(0)
[1] 0.3989423
> dnorm(-2)
[1] 0.05399097
> dnorm(-4,0,2)
[1] 0.02699548
```

pmf

```
> dpois(5,10)
[1] 0.03783327
> dpois(10,5)
[1] 0.01813279
> dpois(5,2.7)
[1] 0.08036047
> dpois(2.7,5)
[1] 0
Warning message:
In dpois(2.7, 5) : non-integer x = 2.700000
> dpois(2,5)
[1] 0.08422434
```

quantiles

```
> qnorm(0.5)
[1] 0
> qnorm(0.02275013)
[1] -2
> qnorm(0.02275013,0,2)
[1] -4
```

```
> qpois(.0671,10)
[1] 6
> qpois(.067,10)
[1] 5
```

Vectors

```
> v=c(1, 2, 3, 4, 5)
```

same result: `v=c(1:5)`

also: `v=1:5`

```
> is.vector(v)
[1] TRUE
> length(v)
[1] 5
> w=300:320
> w
 [1] 300 301 302 303 304 305 306 307 308 309
[11] 310 311 312 313 314 315 316 317 318 319
[21] 320
> w[21]
[1] 320
> w[22]
[1] NA
> w[21, 14, 7]
Error in w[21, 14, 7] : incorrect number of dimensions
> w[c(21, 14, 7)]
[1] 320 313 306
```

What is the point of "[1]"?

What happens if you use an index that is not there?

Rearranging vectors

```
> x=c(1, 2, 2, 9, 0, -3, -5, 0.4, 11)
> sort(x)
[1] -5.0 -3.0  0.0  0.4  1.0  2.0  2.0  9.0
[9] 11.0
```

Assigns ranks, notice management of ties

```
> rank(x)
[1] 5.0 6.5 6.5 8.0 3.0 2.0 1.0 4.0 9.0
> order(x)
[1] 7 6 5 8 1 2 3 4 9
```

Returns a permutation ordering them (ascending)

Look up the help files on these to get more details! E.g.: `> help(order)`

Rearranging vectors

```
> x=c(1, 2, 2, 9, 0, -3, -5, 0.4, 11)
> sort(x)
[1] -5.0 -3.0  0.0  0.4  1.0  2.0  2.0  9.0
[9] 11.0
> rank(x)
[1] 5.0 6.5 6.5 8.0 3.0 2.0 1.0 4.0 9.0
> order(x)
[1] 7 6 5 8 1 2 3 4 9
```

Assigns ranks, notice management of ties

Returns a permutation ordering them (ascending)

Look up the help files on these to get more details! E.g.: `> help(order)`

```
> sort(x, decreasing = TRUE)
[1] 11.0  9.0  2.0  2.0  1.0  0.4  0.0 -3.0 -5.0
```

How can x[order(…)] be expressed more easily?

```
> x[order(x)]
[1] -5.0 -3.0  0.0  0.4  1.0  2.0  2.0  9.0  11.0
> sort(x)
[1] -5.0 -3.0  0.0  0.4  1.0  2.0  2.0  9.0  11.0
```

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? 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 `NA`s. If `TRUE`, missing values in the data are put last; if `FALSE`, they are put first; if `NA`, they are removed.
`partial`	`NULL` or a vector of indices for partial sorting.

Strings/vectors with characters

```
> letters[1:30]
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
[11] "k" "l" "m" "n" "o" "p" "q" "r" "s" "t"
[21] "u" "v" "w" "x" "y" "z" NA  NA  NA  NA
> class(letters[1:26])
[1] “character"

> S="Why do I like R?”
[1] “character"
> substr(S,8,15)
[1] "I like R"
```

Look up the help file for `substr()` and learn also about more functions operating on text mentioned there: `strspilit(), paste(), nchar()`

Create vectors:

- Create a vector with 10 Poisson(3) distributed values.
- Create a vector `prime` containing all one digit prime numbers starting from 2.
- Without typing it into R, say what will > `prime[order(-prime)]` show?
- Without typing it into R, say what will the following commands show?
  `> c(1:10)[prime]  > c(-10:5[prime])  > c(-5:9)[order(-prime)]`

- Create a vector `u` containing all odd numbers smaller than 26 starting from 1. Do so without listing all these numbers.
- Try what happens if you type `LETTER[1]`
- Print every other capital letter starting with “A”. Make the command as short as possible. You may use `u.`
- Print every other letter starting with “B”. Make the command as short as possible. You may use `u.`
- Print every other letter starting with “Z”going backwards. Make the command as short as possible. You may use `u.`

Vector arithmetic and functions

```
> z=seq(0,1,.1)          same result:   > z=(0:10)/10
> z
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
> z*5                    other operator: + - /
 [1] 0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
> z+(0:10)
 [1]  0.0  1.1  2.2  3.3  4.4  5.5  6.6  7.7  8.8  9.9 11.0
> z^2
 [1] 0.00 0.01 0.04 0.09 0.16 0.25 0.36 0.49 0.64 0.81 1.00
> cos(z)
 [1] 1.0000000 0.9950042 0.9800666 0.9553365 0.9210610 0.8775826
 [7] 0.8253356 0.7648422 0.6967067 0.6216100 0.5403023
```

More functions operating on vectors:

```
sin(), tan(), atan(), exp(), log(), log2(), cosh(), sinh(),
round(,), mean(), sd(), sum(), min(), max(), var(), range(),
choose(,), factorial()
```

```
> z
> z+(1:2)  [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
 [1] 1.0 2.1 1.2 2.3 1.4 2.5 1.6 2.7 1.8 2.9 2.0
Warning message:
In z + (1:2) :
  longer object length is not a multiple of shorter object length

> (1:10)+(1:2)
 [1]  2  4  4  6  6  8  8 10 10 12
```

The re-use of the values in the shorter vector is called recycling!

```
> z<0.7
 [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE

> z<0.7|z>0.9
 [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE
TRUE

> 2^(1:10)
 [1]    2    4    8   16   32   64  128  256  512 1024
```

Indexing vectors

```
> x=c(1,-1,7,7,9,1,0)
> x[3]
[1] 7
> x[3:5]
[1] 7 7 9
> x[-5]
[1]  1 -1  7  7  1  0
> x[x>2]
```

`-5` means the 5th component is omitted!

```
> x>2
 [1] FALSE FALSE  TRUE  TRUE  TRUE FALSE FALSE
```

`TRUE` means you include this component,

`FALSE` means you don't

`1,-1,7,7,9,1,0`

Indexing vectors

```
> x=c(1,-1,7,7,9,1,0)
> x[3]
[1] 7
> x[3:5]
[1] 7 7 9
> x[-5]
[1]  1 -1  7  7  1  0
> x[x>2]
[1] 7 7 9
> y=x^x
> y
[1]       1      -1  823543  823543 387420489       1       1
> x[y==1]
[1] 1 1 0
> which(y==1)
[1] 1 6 7
```

Operations with characters

```
> names=c('Emily','Li Min','Pietro','Jose')
> sex=c('F','F','M','M')
> nation=c('UK','China','Italy','Mex')
> names[sex=='F']
[1] "Emily"   "Li Min"
> names[sex=='M'&!nation=='Mex']
[1] "Pietro"
> names[names>'K' | nation=='UK']
[1] "Emily"   "Li Min" "Pietro"

> c(names,'Zarathustra')
[1] "Emily" "Li Min" "Pietro" "Jose" "Zarathustra"
```

Lists

```
> L=list(name=c('Thomas Bayes', 'Maryam Mirzakhani', 'Gottfried
Wilhelm Leibniz'), subject=c('probability', 'geometry', 'analysis'),
born=c(1701, 1977, 1646))
> L$name
[1] "Thomas Bayes"  "Maryam Mirzakhani"  "Gottfried Wilhelm Leibniz"
> L$subject
[1] "probability" "geometry"    "analysis"
> L$born
[1] 1701 1977 1646
> L[[3]]
[1] 1701 1977 1646
```

Lists

```
> L=list(name=c('Thomas Bayes', 'Maryam Mirzakhani', 'Gottfried
Wilhelm Leibniz'), subject=c('probability', 'geometry', 'analysis'),
born=c(1701, 1977, 1646))
> L$name
[1] "Thomas Bayes"  "Maryam Mirzakhani"  "Gottfried Wilhelm Leibniz"
> L$subject
[1] "probability" "geometry"    "analysis"
> L$born
[1] 1701 1977 1646
> L[[3]]
[1] 1701 1977 1646
> L
$name
[1] "Thomas Bayes"  "Maryam Mirzakhani"  "Gottfried Wilhelm Leibniz"

$subject
[1] "probability" "geometry"   "analysis"

$born
[1] 1701 1977 1646
```

Example code to try at home

```
> class(2)
[1] "numeric"
> class(2:3)
[1] "integer"
> class(2L)
[1] "integer"
> class(2+3i)
[1] "complex"
> class(TRUE)
[1] "logical"
> as.complex(2)
[1] 2+0i
> class('C')
[1] "character"
> class('2.2')
[1] "character"
> as.numeric('2.2')
[1] 2.2
> as.integer('2.2')
[1] 2
> as.complex(2)
[1] 2+0i
```

```
> x=(1:3)
> y=x[4]
> y
[1] NA
> class(y)
[1] "integer"
> x=(1:3)+.5
> y=x[4]
> y
[1] NA
> class(y)
[1] “numeric"
```

```
> x=list(1,2,3,'A')
> class(x)
[1] “list"
> x
[[1]]
[1] 1

[[2]]
[1] 2

[[3]]
[1] 3

[[4]]
[1] "A"
```

Operating with vectors and lists

- Create the vector (1,2,3,4,5) in R, call it `v`, and divide each component by -1. Can you do this with a command that is at most 5 characters long?
- Divide the components of `v` by 0.1,0.2,0.3,0.4,0.5, respectively. Do so with a command that is at most 6 characters long.
- Without typing it into R, what is the output of the following command?
  `> (v-3)^2 != 0`
- First generate a vector `w` of length 20 with entries of your choice. Then remove the components in the even positions. Use no more than 15 characters to create this expression. (You don't have to create any variable for the vector you output.)
- You type `> seq(0,1,.1)=0.5` and get the error message below. Explain what it actually means and what the correct command should be.
  `target of assignment expands to non-language object`
- Create three vectors of lengths three. The first one contains three items, the second one has the colours, and the third one gives their weight. Write some commands that select objects based on their colour or their weight. Also try to select them based on a combination of conditions about colour and weight.
- Do the previous exercise again, but use the data structure list.

Arrays and matrices

```
> matrix(1:12,3,4)
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> matrix(1:12,4,3)
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
> matrix(1:12,4,3,byrow=TRUE)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12

> x=round(matrix(rnorm(9),3,3),2)
> x
      [,1]  [,2]  [,3]
[1,]  0.41 -0.47  2.97
[2,] -0.25 -0.07 -1.15
[3,]  0.71  0.08 -0.02
```

```
> diag(x)
[1]  0.41 -0.07 -0.02

> diag(x)=c(1,2,3)
> x
      [,1]  [,2]  [,3]
[1,]  1.00 -0.47  2.97
[2,] -0.25  2.00 -1.15
[3,]  0.71  0.08  3.00

> diag(c(1,2,3),nrow=3,ncol=5)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    0    0    0    0
[2,]    0    2    0    0    0
[3,]    0    0    3    0    0

> class(x)
[1] "matrix"
```

solve() inverts matrices

```
> m=solve(x)
> m
             [,1]       [,2]       [,3]
[1,]  3.29929514  0.8923044 -2.9242521
[2,] -0.03601496  0.4827088  0.2206932
[3,] -0.77987278 -0.2240510  1.0195212
```

```
> y=matrix(1:6,2,3)
> y
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> dim(y)
[1] 2 3
> rbind(x,y)
      [,1]  [,2]  [,3]
[1,]  1.00 -0.47  2.97
[2,] -0.25  2.00 -1.15
[3,]  0.71  0.08  3.00
[4,]  1.00  3.00  5.00
[5,]  2.00  4.00  6.00
> cbind(x,t(y))
      [,1]  [,2]  [,3] [,4] [,5]
[1,]  1.00 -0.47  2.97    1    2
[2,] -0.25  2.00 -1.15    3    4
[3,]  0.71  0.08  3.00    5    6
```

actions on matrices are component wise

```
> x+x
      [,1]  [,2]  [,3]
[1,]  2.00 -0.94  5.94
[2,] -0.50  4.00 -2.30
[3,]  1.42  0.16  6.00

> x*x
       [,1]   [,2]   [,3]
[1,] 1.0000 0.2209 8.8209
[2,] 0.0625 4.0000 1.3225
[3,] 0.5041 0.0064 9.0000
```

```
> x%*%y
Error in x %*% y : non-conformable arguments
> x%*%t(y)
      [,1]  [,2]
[1,] 14.44 17.94
[2,]  0.00  0.60
[3,] 15.95 19.74
> x*t(y)
Error in x * t(y) : non-conformable arrays
          > x[1,]
          [1]  1.00 -0.47  2.97
          > x[,1]
          [1]  1.00 -0.25  0.71
> x[1:2,]
      [,1]  [,2]  [,3]
[1,]  1.00 -0.47  2.97
[2,] -0.25  2.00 -1.15
> x[1:2,1:2]
      [,1]  [,2]
[1,]  1.00 -0.47
[2,] -0.25  2.00
```

```
> eigen(x)
$values
[1] 3.7884132 1.9633386 0.2482482

$vectors
           [,1]        [,2]       [,3]
[1,]  0.6823538 -0.22643336  0.9681554
[2,] -0.4604741  0.97072168 -0.0253376
[3,]  0.5677648  0.08017078 -0.2490644
```

Arrays

2-dimensional arrays are like matrices, though created with slightly different syntax.

```
> r=runif(4)
> r
[1] 0.3722298 0.5411991 0.5813346 0.2281324
> x=matrix(r,2,2)
> x
          [,1]      [,2]
[1,] 0.3722298 0.5813346
[2,] 0.5411991 0.2281324
> x=matrix(r,2)
> x
          [,1]      [,2]
[1,] 0.3722298 0.5813346
[2,] 0.5411991 0.2281324
> y=array(r,c(2,2))
> y
          [,1]      [,2]
[1,] 0.3722298 0.5813346
[2,] 0.5411991 0.2281324
> class(x)
[1] "matrix"
> class(y)
[1] "matrix"
```

runif() creates random numbers

```
> y=array(runif(12),c(2,3,2))
> y
, , 1

          [,1]      [,2]      [,3]
[1,] 0.1019307 0.1391619 0.2425620
[2,] 0.6997855 0.9238783 0.8779895

, , 2

             [,1]      [,2]      [,3]
[1,] 0.2321210748 0.0433073 0.2444532
[2,] 0.0003120764 0.8428333 0.9978538

> class(y)
[1] "array"
> dim(y)
[1] 2 3 2
```

Create matrices and arrays:

- Create a 4 by 5 array using 20 values sampled from a normal distribution. Display the last 3 columns. Display the array without the first row.
- Create the same but using matrix syntax.
- Create the array using only three normally distributed values by recycling them.
- Create a 3 by 4 array using the first 12 letters of the alphabet. Display the first two rows.
- Create a 3-dimensional array of dimensions 2,3,4 and display it by showing two subarrays.

R object classes

- Object-oriented: Objects (containing data) and methods (that act on them). How a method works depends on the kind of object.
- Two basic data types in R
  - Atomic vectors: Array of same kind of value
  - Lists: Collections of potentially different data types