Data Programming with R

Isabella Gollini

Lecture 7 - Basic mathematics and statistics in R

Basic mathematics and statistics in R

- Maths functions
  - calculus
  - sorting
  - set operations
- Linear Algebra in R
  - solving matrices
  - matrix decomposition
  - `diag`
  - `sweep`
- Basic statistics in R
  - correlation
  - t-tests
  - linear regression
- Examples

Basic Maths

- We have met most simple arithmetic operations in R: `+`, `-`, etc.
- R has many more (all vectorised):
  - `exp()`: exponential function
  - `log()`: natural logarithm (see also `log10` and `log2`)
  - `sqrt()`: square root
  - `abs()`: absolute value
  - `sin()`, `cos()`, `tan()`, `asin()`, `acos()`,...: trigonometry
  - `min()`, `max()`, `pmin()`, `pmax()`, `which.min()`, `which.max()`: minima and maxima
  - `sum()`, `prod()`, `cumsum()`, `cumprod()`: sums and products
  - `round()`, `floor()`, `ceiling()`, `signif()`: methods for rounding numbers

`min`, `max`, `pmin`, and `pmax`.

`min` and `max` are quite straightforward:

```r
z <- matrix(c(1, 5, 6, 2, 3, 2), nrow = 3, ncol = 2)
z
```

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

```r
min(z)
```

```
## [1] 1
```

```r
z1 <- z[, 1]
z2 <- z[, 2]
min(z1, z2)
```

```
## [1] 1
```

```r
pmin(z1, z2)
```

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

- `min` and `max` just concatenate everything into a vector and find the min or max.
- `pmin` and `pmax` find parallel minima/maxima across multiple vectors or matrices.

Calculus

R can do symbolic differentiation and numerical integration:

```r
D(expression(exp(x^2)), 'x')
```

```
## exp(x^2) * (2 * x)
```

```r
integrate(function(x) x^2, 0, 1)
```

```
## 0.3333333 with absolute error < 3.7e-15
```

- Here `D` finds the derivative of a mathematical expression (created by the function `expression`) with respect to the chosen variable.
- The `integrate` function will calculate numerical integrals over the provided range.
- There are many other (more advanced) R calculus packages.

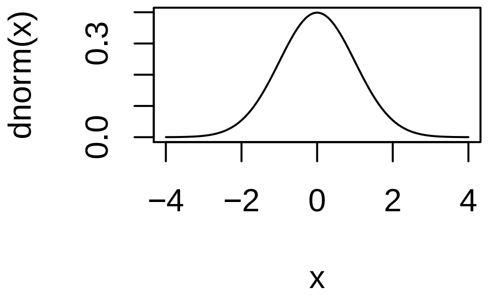
Probability distributions

SC 1

- R contains a large range of functions for evaluating probability distributions. These all follow a set pattern:
  - `d` for density (or pdf)
  - `p` for cumulative density (or cdf)
  - `q` for quantiles
  - `r` for random number generation
- Distributions include: `norm` (Normal), `binom` (Binomial), `chisq` (Chi-squared), `f` (F-distribution), `pois` (Poisson), `t`, etc, etc

Example: plotting the pdf of a normal distribution:

```r
x <- seq(-4, 4, length = 100)
plot(x, dnorm(x), type = 'l')
```



Sorting

R has a few different functions for sorting data. We will cover `sort`, `order` and `rank`.

```r
x <- c(5, 13, 12, 5)
sort(x)
```

```
## [1]  5  5 12 13
```

```r
order(x)
```

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

```r
rank(x)
```

```
## [1] 1.5 4.0 3.0 1.5
```

- `sort` takes a vector and puts it in order.
- `order` gives the order of the vector `x` (i.e. the first element is the smallest, etc).
- `rank` gives the rank order of the vector (so the 1st and 4th elements are the smallest and are given rank (1 + 2) / 2.
- Both `sort` and `order` have an extra argument `decreasing` for reversing the operation.

More sorting

If you want to sort a matrix by one of the rows, the easiest way is to use `order`:

```r
d <- data.frame(kids = c('Jack', 'Jill', 'Billy'), ages = c(12, 10, 13))
d[order(d$kids),]
```

```
##    kids ages
## 3 Billy   13
## 1  Jack   12
## 2  Jill   10
```

```r
d[order(d$ages),]
```

```
##    kids ages
## 2  Jill   10
## 1  Jack   12
## 3 Billy   13
```

Set operations

- If you have a set of items stored in a vector, R can compare them using set operations:
  - `union(x, y)`: Creates the union set of `x` and `y`
  - `intersect(x, y)`: Creates the intersection of `x` and `y`
  - `setdiff(x, y)`: Creates the set of all objects in `x` that are not in `y`
  - `setequal(x, y)`: Determines if `x` and `y` are equal
  - `c %in% y`: Tests whether `c` is an element of `y`

Set operations: examples

```r
x <- 1:5
y <- 4:8
union(x, y) # Creates the union set of x and y
```

```
## [1] 1 2 3 4 5 6 7 8
```

```r
intersect(x, y) # Creates the intersection of x and y
```

```
## [1] 4 5
```

```r
setdiff(x, y) # Creates the set of all objects in x that are not in y
```

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

```r
setequal(x, y) # Determines if x and y are equal
```

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

```r
4 %in% y # Tests whether 4 is an element of y
```

```
## [1] TRUE
```

Linear Algebra in R

- Solving matrices
- matrix decomposition
- `diag`
- `sweep`

Linear Algebra

- We have already met vector operations and matrix multiplication (`%*%`).
- There are other useful linear algebra functions:
  - `solve()`: finds a matrix equation solution or matrix inverse
  - `t()`: matrix transpose
  - `chol()`, `qr()`, `eigen()`: Cholesky, QR and eigen decomposition
  - `det()`: Determinant
  - `diag()`: Find or set the matrix diagonal
  - `sweep()`: Removes summary statistics from matrices.
- You should be reasonably familiar with these terms.
- See the screencast for a more in-depth look.

Matrix `solve`

`solve` is useful for finding solutions to simultaneous equations:

$$\begin{aligned} x_1 + x_2 &= 2 \qquad (1) \\ -x_1 + x_2 &= 4 \qquad (2) \end{aligned}$$

We can re-write this in matrix form:

$$\begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \end{bmatrix}$$

$$Ax = b$$

```r
A <- matrix(c(1, -1, 1, 1), 2, 2)
b <- c(2, 4)
x <- solve(A, b)
x
```

```
## [1] -1  3
```

Matrix decomposition

- Decomposing a matrix is a bit like finding the prime factors of a number, e.g. $6 = 3 \times 2$.
- When dealing with a square matrix A there are multiple ways of splitting it up:
  - $A = QR$ where Q is an orthogonal matrix and R is upper triangular.
  - $A = LL^T$ where L is a lower triangular matrix.
  - $A = VDV^{-1}$ where D is a diagonal matrix of eigenvalues and V is a matrix of eigenvectors.

```r
M2 <- matrix(c(3, 1, 1, 3),
  2, 2)
U <- chol(M2) # upper triangular matrix
t(U) %*% U
```

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

```
eigenM2 <- eigen(M2)
D <- diag(eigenM2$values)
V <- eigenM2$vectors
V %*% D %*% solve(V)
```

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

Save time using parentheses!

- Clever use of parentheses makes your code running much faster!
- Use the properties of matrix multiplication:
  - Associative property of multiplication: $(AB)C = A(BC)$
  - Distributive properties $A(B + C) = AB + AC$

```
n <- 2500
m <- 10
A <- matrix(n * n, nrow = n, ncol = n)
B <- matrix(n * n, nrow = n, ncol = n)
C <- matrix(n * m, nrow = n, ncol = m)

system.time(A %*% B %*% C)
```

```
##    user  system elapsed
##  22.306   0.187  26.997
```

```
system.time(A %*% (B %*% C))
```

```
##    user  system elapsed
##    0.19    0.00    0.20
```

- The function `system.time` estimates the amount of time a particular command or function takes to run.

`diag`

- `diag` works to obtain the diagonal from a matrix, to set it, or to create a diagonal matrix:

```
M <- matrix(1:9, 3, 3)
diag(M)
```

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

```
diag(M) <- 7
M
```

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

```
M <- diag(c(3, 1))
M
```

```
##      [,1] [,2]
## [1,]    3    0
## [2,]    0    1
```

sweep function

sweep is a neat function to remove a summary statistic from a numeric data frame/matrix:

```
head(swiss, 3) # Swiss Fertility and Socioeconomic Indicators (1888) Data
```

```
##              Fertility Agriculture Examination Education Catholic
## Courtelary        80.2        17.0          15        12     9.96
## Delemont          83.1        45.1           6         9    84.84
## Franches-Mnt      92.5        39.7           5         5    93.40
##              Infant.Mortality
## Courtelary               22.2
## Delemont                 22.2
## Franches-Mnt             20.2
```

```
# centre the variables around their mean
mean.swiss <- apply(swiss, 2, mean) # calculate the column means. Same as colMeans(swiss)
swiss.mean0 <- sweep(swiss, 2, mean.swiss)
head(swiss.mean0, 3)
```

```
##              Fertility Agriculture Examination Education  Catholic
## Courtelary    10.05745  -33.659574   -1.489362  1.021277 -31.18383
## Delemont      12.95745   -5.559574  -10.489362 -1.978723  43.69617
## Franches-Mnt  22.35745  -10.959574  -11.489362 -5.978723  52.25617
##              Infant.Mortality
## Courtelary          2.2574468
## Delemont            2.2574468
## Franches-Mnt        0.2574468
```

```
# standardise the variables z = (x - mean) / sd
sd.swiss <- apply(swiss, 2, sd) # # calculate the column standard deviations
swiss.standardised <- sweep(sweep(swiss, 2, mean.swiss), 2, sd.swiss, "/")
head(swiss.standardised, 3)
```

```
##              Fertility Agriculture Examination  Education   Catholic
## Courtelary   0.8051305  -1.4820682  -0.1866863  0.1062125 -0.7477267
## Delemont     1.0372847  -0.2447942  -1.3148051 -0.2057867  1.0477479
## Franches-Mnt 1.7897846  -0.4825622  -1.4401516 -0.6217858  1.2529998
##              Infant.Mortality
## Courtelary         0.77503669
## Delemont           0.77503669
## Franches-Mnt       0.08838778
```

Basic Statistics in R

- Correlation
- t-tests
- linear regression

Basic statistics in R

- We will cover 3 types of basic statistical analysis you might like to run:
  - Correlation
  - t-tests
  - Linear regression

- You can learn about implementing some of the more advanced statistical modelling topics, including generalised linear modelling (GLMs), basic multivariate analysis, and time series analysis, in R either from the recommended texts, or from some of the other STAT modules we offer.

Correlation

- Correlation measures the degree of relatedness between two (or more) vectors. The Pearson correlation is defined as:
$$\rho = \frac{\text{Cov}(x, y)}{\text{sd}(x)\ \text{sd}(y)}$$
- The R functoin is `cor`:

```r
cor(swiss[,1:3])
```

```
##              Fertility Agriculture Examination
## Fertility    1.0000000   0.3530792  -0.6458827
## Agriculture  0.3530792   1.0000000  -0.6865422
## Examination -0.6458827  -0.6865422   1.0000000
```

- An extra argument (`method`) allows you to change to Spearman or Kendall correlation.
- `cor.test` will run a hypothesis test on two vectors to test for statistically significant association (though be aware this assumes the data are bivariate normal).

t-test

- Usual set-up: 2 vectors (can be different lengths) where you want to test whether or not they have the same mean.
- Can also have situations where data are paired (eg. two measurements on same person) or for a single vector where you want to test the mean is a certain value.
- R function is `t.test`:

```r
t.test(extra ~ group, data = sleep, paired = TRUE)
```

```
##
##  Paired t-test
##
## data:  extra by group
## t = -4.0621, df = 9, p-value = 0.002833
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -2.4598858 -0.7001142
## sample estimates:
## mean of the differences
##                   -1.58
```

- The `sleep` dataset contains data which show the effect of two soporific drugs (increase in hours of sleep compared to control) on 10 patients.

Linear regression

- Setup: have a response variable (y) and (possibly multiple) explanatory variables ($x_1, \dots, x_p$). Want to fit the model:
$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_p x_p + \epsilon \epsilon \sim N(0, \sigma^2)$$
- We can fit this using maximum likelihood.

Linear regression

R function is `lm()` where the first argument is a formula:

```
mod <- lm(Fertility ~ ., data = swiss)
summary(mod)
```

```
##
## Call:
## lm(formula = Fertility ~ ., data = swiss)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.2743  -5.2617   0.5032   4.1198  15.3213
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)      66.91518   10.70604   6.250 1.91e-07 ***
## Agriculture      -0.17211    0.07030  -2.448  0.01873 *
## Examination      -0.25801    0.25388  -1.016  0.31546
## Education        -0.87094    0.18303  -4.758 2.43e-05 ***
## Catholic          0.10412    0.03526   2.953  0.00519 **
## Infant.Mortality  1.07705    0.38172   2.822  0.00734 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.165 on 41 degrees of freedom
## Multiple R-squared:  0.7067, Adjusted R-squared:  0.671
## F-statistic: 19.76 on 5 and 41 DF,  p-value: 5.594e-10
```

Extended examples

- Calculating probabilities
- snap
- Fitting logistic regression models

Calculating probabilities

- The maths behind many probabilities can be very hard to calculate algebraically. It is often much easier to simulate the events and work out how frequently they occur.
- In this section we're going to work out some uncertain quantities:
  - The probability of getting at least 4 heads from 5 tosses of a coin.
  - The expected value of the maximum of two $N(0, 1)$ variables.
  - The expected number of draws before identical playing cards appear in a game of snap.

Heads and Tails

- What is the probability of getting at least 4 heads from 5 tosses of a coin?
- We could do this algebraically using the probability mass function of the Binomial distribution. ($1 =$ head, $0 =$ tail).

```
1 - pbinom(3, 5, 0.5) # Pr(X >= 4) = 1 - Pr(X <= 3)
```

```
## [1] 0.1875
```

```
# or
pbinom(3, 5, 0.5, lower.tail = FALSE) # Pr(X >= 4) = Pr(X > 3)
```

```
## [1] 0.1875
```

- Alternatively we could simulate a large number of such coin tosses and find how often at least 4 heads occur:

```
large <- 100000
x <- rbinom(large, 5, 0.5)
mean(x >= 4)
```

```
## [1] 0.18814
```

- Remember `x >= 4` is a logical variable so finding the mean is equivalent to finding the proportion of times such an event occurs.

Maxima of two N(0,1)

- Suppose we want to find the mean of the maximum of two standard normal random variables.

This is challenging to work out algebraically… but it's pretty quick to work out (approximately) on a computer.

```
nreps <- 100000
sum <- 0
for(i in 1:nreps) {
  xy <- rnorm(2)
  sum <- sum + max(xy)
  }
print(sum / nreps)
```

```
## [1] 0.5643907
```

- We can also run the above without any loops:

```
x <- rnorm(nreps)
y <- rnorm(nreps)
maxxy <- pmax(x, y)
mean(maxxy)
```

```
## [1] 0.5624389
```

Snap!

- Suppose you play a game of snap with two players who each have a shuffled pack of cards. You both turn over cards until you reach an identical pair.
- How many cards, on average, would you have to turn over before the game ends?

```
cards <- 1:52
p1hand <- sample(cards)
p2hand <- sample(cards)
snaptime <- which((p1hand - p2hand == 0) == TRUE)[1]
snaptime
```

```
## [1] 24
```

- Here it took 24 draws before snap was called

More hands of Snap!

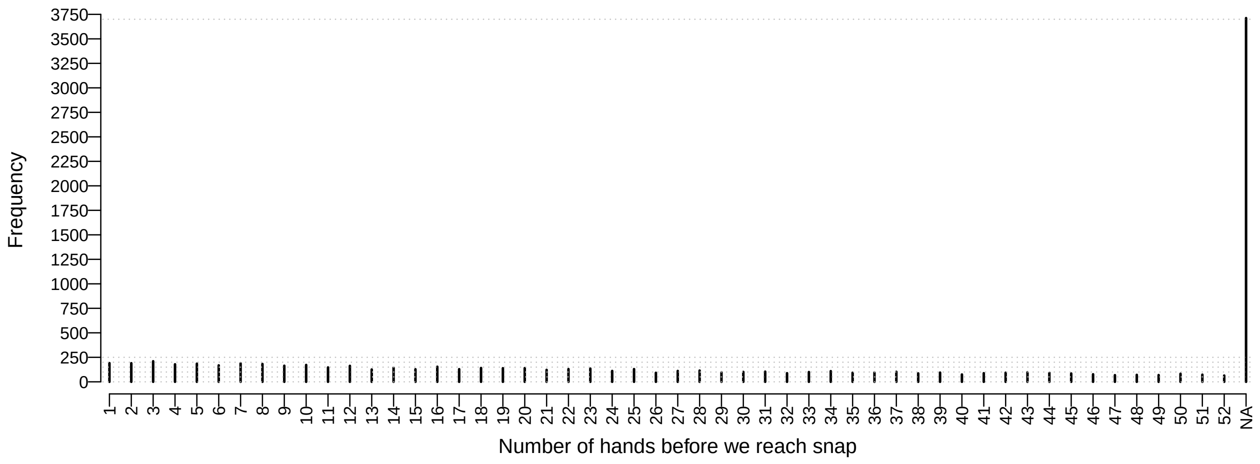
Now let's try working out the value with a loop:

```
numhands <- 100
snaptimes <- vector(length = numhands)
for(i in 1:numhands) {
  p1hand <- sample(cards)
  p2hand <- sample(cards)
  snaptimes[i] <- which((p1hand - p2hand == 0) == TRUE)[1]
}
```

and without a loop:

```
snap <- function(handdiff) which((handdiff == 0) == TRUE)[1]
# Create lots of hands and then run lots of applys
numhands <- 10000
bighand1 <- apply(matrix(rep(cards, numhands), 52, numhands), 2, sample)
bighand2 <- apply(matrix(rep(cards, numhands), 52, numhands), 2, sample)
snaptimes2 <- apply(bighand1 - bighand2, 2, snap)
```

More hands of Snap!



A good estimate of the number of hands before we reach snap is the number of times it took divided by the number of times you got snap

```
(sum(snaptimes2, na.rm = TRUE) + sum(52 * is.na(snaptimes2))) / sum(!is.na(snaptimes2))
```

```
## [1] 52.66444
```

Fitting logistic regression models

Back in lecture 3 we fitted a logistic regression model to the birth weight data. We used the formula:

$$P(Y = 1|X = x) = \frac{1}{1 + \exp[-(\beta_0 + \beta_1 x)]}$$

The similar formula for $Y = 0$ is:

$$P(Y = 0|X = x) = 1 - \frac{1}{1 + \exp[-(\beta_0 + \beta_1 x)]}$$

A common way of finding the values of the missing parameters is to use maximum likelihood, i.e. maximise the probability of the data:

$$L(\beta_0, \beta_1; y) = \prod_{i=1}^{n} P(Y = y_i|X = x_i)$$

$$l(\beta_0, \beta_1; y) = \log L(\beta_0, \beta_1; y) = \sum_{i=1}^{n} \log P(Y = y_i|X = x_i)$$

A function for calculating the likelihood

Here's a function to calculate the negative of the log-likelihood for any given values of the parameters:

```r
neglikfun <- function(parameters) {
  beta0 <- parameters[1]
  beta1 <- parameters[2]
  # separate data in 0s and 1s
  daty0 <- y == 0
  daty1 <- y == 1
  # calculate the log-likelihood
  loglikelihood <- sum(log(1 / (1 + exp(- beta0 - beta1 * x[daty1])))) +
    sum(log(1 - (1 / (1 + exp(- beta0 - beta1 * x[daty0])))))
  # return the negative of the log-likelihood
  return(-loglikelihood)
}
```

- This function does not define `y` or `x` in the arguments. They're going to be treated as global variables.

Computing the negative log-likelihood

We can now call the function to compute the negative log-likelihood.

```r
library(MASS)
data(birthwt)
x <- birthwt$age
y <- birthwt$low
neglikfun(c(1, -0.1))
```

```
## [1] 121.2885
```

- We want to find the parameter values that minimise this function.
- We're going to run this through an optimisation routine. R's optimisation routines in general find minima rather than maxima so we're going to minimise the negative log-likelihood.

Minimising the negative log-likelihood

- R has a few different functions for optimisation. We will use `optim`.
- The usual syntax for `optim` is `optim(par, fun)` where `par` is an initial guess at the parameters and `fun` is a function which takes only the parameters to be optimised as arguments.

We can run it for our example to get:

```r
res <- optim(c(0, 0), neglikfun)
res
```

```
## $par
## [1]  0.38517136 -0.05117567
##
## $value
## [1] 115.956
##
## $counts
## function gradient
##       67       NA
##
## $convergence
## [1] 0
##
```

```
## $message
## NULL
```

- This gives us the optimum values of the parameters, the value of the (negative) log-likelihood for the optimum and a measure of whether it has converged (`0 = yes`).

Did it work?

We can compare our results with our output from the `glm` function:

```
mod <- glm(low ~ age, family = binomial, data = birthwt)
mod$coefficients
```

```
## (Intercept)         age
##  0.38458192 -0.05115294
```

```
logLik(mod) # Function to extract the Log-Likelihood from a "glm" object
```

```
## 'log Lik.' -115.956 (df=2)
```

```
# Our results:
res$par
```

```
## [1]  0.38517136 -0.05117567
```

```
- res$value
```

```
## [1] -115.956
```

- We have approximately the same values!
- Some extra notes:
  - The reason our function works with `optim` is that `neglikfun` only takes parameters as an argument. If we had included `x` and `y` `optim` would have failed.
  - The `glm` function creates lots of extra useful output, involving far more than just maximum likelihood estimates (MLEs).

Lessons from this week

- Lots of useful maths function in R: calculus, sorting, matrix manipulation.

- Probability distributions pre.fixed with `d`, `p`, `q`, or `r`.

- Now know how to run some simple statistical analyses in R.