Practice R

Justas Mundeikis
Data-Science.lt

2020-07-21

1 Introduction

...

2 Data structures

2.1 Vectors

A vector is the most basic data structure in R. It contains only elements of the same type: * character elements (“name”, “grade” “a”,“b”, “c”) * integer elements (1, 2, 3L) * numerical elements (1.1, -0.1,9.999) * booleen elements (TRUE, FALSE) * complex (1+i) * raw ()

Vectors are usually created using the function `c()` [=`concatenate`]. Vectors can be assigned in multiple ways to an object, either by using `<-` operator, by using `=` operator (not suggested to use this), or by using the `assign()` function.

The type of vector can allways be checked using the function: `typeof()`. The number of elements of a vector can be checked using the function `length()`

A vector consisting of multiple vectors containing the same type of elements can also be created using `c()`.

```
# Example
## creating vectors
x <- c(1.1,2.2,3.3,4.4,5.5)
y = c("a", "b", "c", "d")
assign("z", c(1,2,3,4,5))

## checking the type of vectors
typeof(x)
## [1] "double"
typeof(y)
## [1] "character"
typeof(z)
## [1] "double"

## checking the number of elements in the vectors
length(x)
## [1] 5
length(y)
## [1] 4
length(z)
## [1] 5

a <- c(1,2,3,4,5)
b <- c(6,7,8,9,10)
c <- c(a,b) #a and b denote object, thus no ".."
print(c)
##  [1]  1  2  3  4  5  6  7  8  9 10
```

2.2 Vector with a sequence of numbers

There are few different ways how to create a vector with a number sequence. The easiest way is to create an integer vector using :

```
# integer sequence from 1 to 10
1:10
##  [1]  1  2  3  4  5  6  7  8  9 10
# is the same as
c(1,2,3,4,5,6,7,8,9,10)
##  [1]  1  2  3  4  5  6  7  8  9 10
```

More advanced way is tu use the function `seq()`. Run the command in the console `?seq` to see the function manual page. Here are some examples of possible `seq()` usage:

```
# read the manual page
?seq

#examples
seq(from=1, to=5)
## [1] 1 2 3 4 5
seq(from=0, to=5, by=0.25 )
##  [1] 0.00 0.25 0.50 0.75 1.00 1.25 1.50 1.75 2.00 2.25 2.50 2.75 3.00 3.25 3.50
## [16] 3.75 4.00 4.25 4.50 4.75 5.00
seq(from=0, to=1, length.out=6)
## [1] 0.0 0.2 0.4 0.6 0.8 1.0
x <- c("a", "b", "c")
seq(along.with=x )
## [1] 1 2 3
seq(from=9) # the same as 1:10, or seq(from=1, to=10)
## [1] 1 2 3 4 5 6 7 8 9
seq(length.out=5.5)
## [1] 1 2 3 4 5 6
```

2.3 Repetition

```
?rep
```

```
x <- 1:5
y <- c("a", "b", "c")
rep(x,times=2)
##  [1] 1 2 3 4 5 1 2 3 4 5
rep(x, each=2)
##  [1] 1 1 2 2 3 3 4 4 5 5
rep(y, length.out=5)
## [1] "a" "b" "c" "a" "b"
```

2.3.1 Questions

1. Create / print a single number vector with the numerical element of `99`
2. Create a numerical vector with even number up to 10.
3. Create a numerical vector with numbers / integers 1 to 50 Explain what do the numbers in square brackets at the beginning of every reported line mean?
4. Concatenate three vectors: a containing integers from 1 to 10, b containing integers from 11 to 20, c containing integers from 21 to 30 by using the function `seq` or `x:x` to a new vector `d`

2.3.2 Solutions

```
# 1.
99
## [1] 99

# 2
c(2,4,6,8,10)
## [1]  2  4  6  8 10

# 3
# first alternative
seq(from=1, to=50)
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
## [26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
# second alternative
1:50
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
## [26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
# third alternative
# c(1,2,3,4,...,100) manual entry by hand

# the number in the brackets at the beginning of each line implicates
# the number of the element in the vector.

# 4
a <- seq(from=1, to=10, by=1)
b <- seq(from=11, to=20, by=1)
c <- seq(from=21, to=30, by=1)
d <- c(a,b,c)
print(d)
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
## [26] 26 27 28 29 30
```

2.4 Vector with a sequance of letters

```
letters
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
## [20] "t" "u" "v" "w" "x" "y" "z"
LETTERS
##  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"
## [20] "T" "U" "V" "W" "X" "Y" "Z"
```

2.5 Vectoriced operations

In R all operations are executed in vectorized form, for example addint two vectors `c(1,2,3)` and `c(4,5,6)` will result in a vector, where each element willl be added elementwise in the vector resutling in `c(5,7,9)`

Basic arithmetic operators are: * `+` addition * `-` subtraction * `*` multiplication * `/` division * `^` or `**` exponentiation * `x %% y` modulus (x mod y) is the integer remainder. usefel for cheking if a number is odd * `x %/% y` integer division

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

a+b
## [1] 4 6 8
a-b
## [1] -2 -2 -2
a*b
## [1]  3  8 15
a/b
## [1] 0.3333333 0.5000000 0.6000000
a^b
## [1]   1  16 243
3 %% 2 # 3 pizza pieces divided by 2 persons, 1 piece remains
## [1] 1
17 %% 2 # if remainder =1, then the number is odd, else even
## [1] 1
b %% a
## [1] 0 0 2
10 %/% 3
## [1] 3
b %/% a
## [1] 3 2 1
```

2.5.1 Exercises

1. Create two vectors a with a sequence of 1 to 5 and b with a sequence of 5 to 10 try all mathematical operators with tese two vectors.
2. Create a new object c by assigning the following arithmetical operation to it using vectors a and b from previous excersise: add to each element of vector `b` 5, then multiple each element by 3, then and divide the reuslting vector by the vector a.
3. Substituting, prove that the square of c which is equals to the sum of vectors a and b is the same as squaring the sum of vectors a and b.

2.5.2 Solutions:

```
a <- seq(from=1, to=5, by=1)
b <- seq(from=6, to=10, by=1)

a+b
## [1]  7  9 11 13 15
a-b
## [1] -5 -5 -5 -5 -5
a*b
## [1]  6 14 24 36 50
a/b
## [1] 0.1666667 0.2857143 0.3750000 0.4444444 0.5000000
a^b
## [1]       1     128    6561  262144 9765625
```

```
b %% a
## [1] 0 1 2 1 0
b %/% a
## [1] 6 3 2 2 2


c <- ((b+5)*3)/a
print(c)
## [1] 33.0 18.0 13.0 10.5  9.0

c <- a+b
c^2
## [1]  49  81 121 169 225
(a+b)^2
## [1]  49  81 121 169 225
# proof
c^2==(a+b)^2
## [1] TRUE TRUE TRUE TRUE TRUE
```

2.6 Vectors to formulas

Mathematical predefined values / formulas:

- `pi`
- `exp()`
- `sqrt()`
- `abs()`
- `log()` see `?log`

It is quite simple using simple assign operations and mathematical operations to translate mathematical formulas.

The radius of a circle is given by a formula

$$Radius = \sqrt{\frac{Area}{\pi}}$$

Lets assume the area is given and equals 30. R knows what π is.

```
# cheking the value for pi
pi
## [1] 3.141593
# defining area
a <- 30
# creating formula for radius
r <- sqrt(a/pi)
# prining the radius resula
print(r)
## [1] 3.090194
```

2.6.1 Exercises:

1. The normally distributed density function has the equation

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

.

Given the mean ($\mu = 0$) and the standard deviation ($\sigma = 1$) calculate the probability of $x = 1.96$

2. Now instead of using single value for `x`, use a vector with a sequence from -5 to 5 by 0.1 steps. Instead of printing the result `print()` use the function `plot(x=x, y=pdf)`.

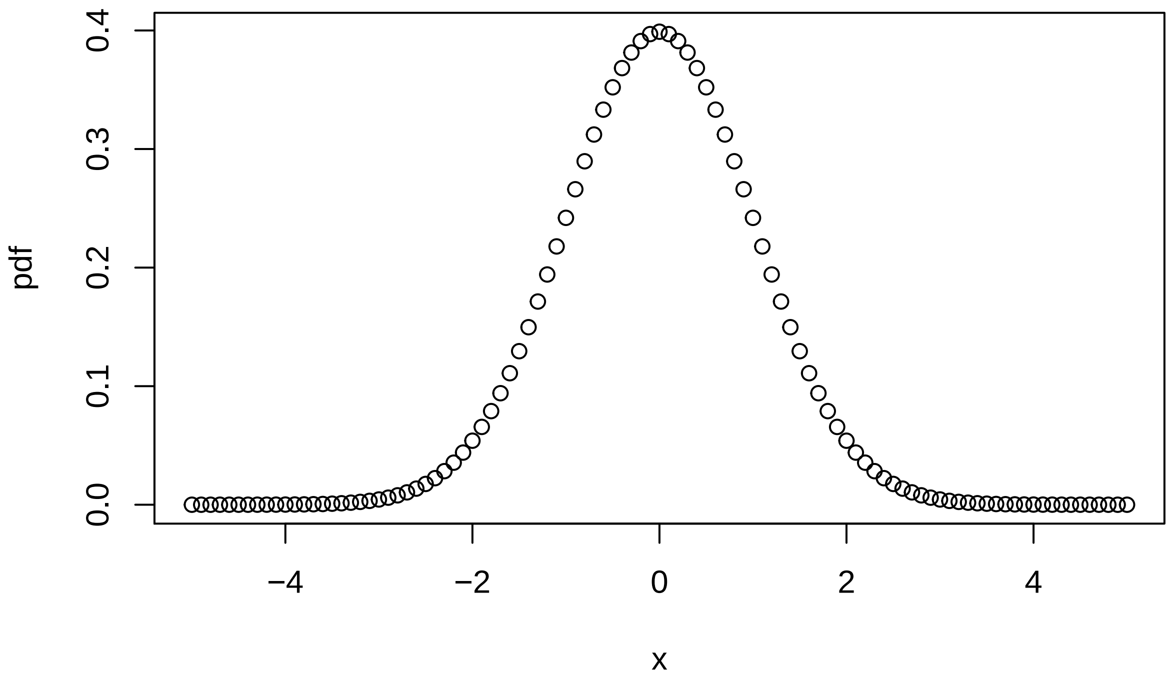
3. Use the modulo operator `%%` to find out for which of the following pairs, the first number is a multiple of the second

- 24025, 115
- 61988, 98
- 2555, 245
- 68719476736, 8

```
# defining  variables
mu <- 0
sigma <- 1
x <- 1.96
#defining formula
pdf <- 1/(sigma*sqrt(2*pi))*exp(-1/2*((x-mu)/sigma)^2)
# print the pdf
print(pdf)
## [1] 0.05844094
```

```
# defining  variables
mu <- 0
sigma <- 1
x <- seq(from=-5, to=5, by=0.1)
```

```r
#defining formula
pdf <- 1/(sigma*sqrt(2*pi))*exp(-1/2*((x-mu)/sigma)^2)
# print the pdf
plot(x=x, y=pdf)
```



```r
24025  %% 115
## [1] 105
61988 %% 98
## [1] 52
2555 %% 245
## [1] 105
68719476736 %% 8
## [1] 0
```

3. You think of taking a mortgage at a bank. The mortgage size (principle - P) equals 100000. Currently the annual interest rate of Euribor equals 6 % (monthly interest rate =0.06/12=0.005). You would like to repay the mortgage in 15 years (15*12=180 months). You would like to know what monthly payments of size M you have to pay. The formula for M is given as

$$M = P\frac{r(1+r)^n}{(1+r)^n - 1}$$

Lets assume you cannot afford to pay the monthly mortgage payment calculcated in first part. So you're interested in calculating diffrent M for different number of periods. Assume you want to calcultae M for periods between 10 and 25 years. Plot the correspinng M values using `plot(x=n, y=M)`

Assume you can pay 300 euro monthly. How many monthly payments have you to pay? Assume you consider different amounts of payments of 100,200,300,400,500,600,700,800,900 euro. Plot the number of months using `plot(x=M, y=n)`

Lets assume you can only repay 650 euro. How many months have you to repay your mortgage? Note , that n can be calculated as

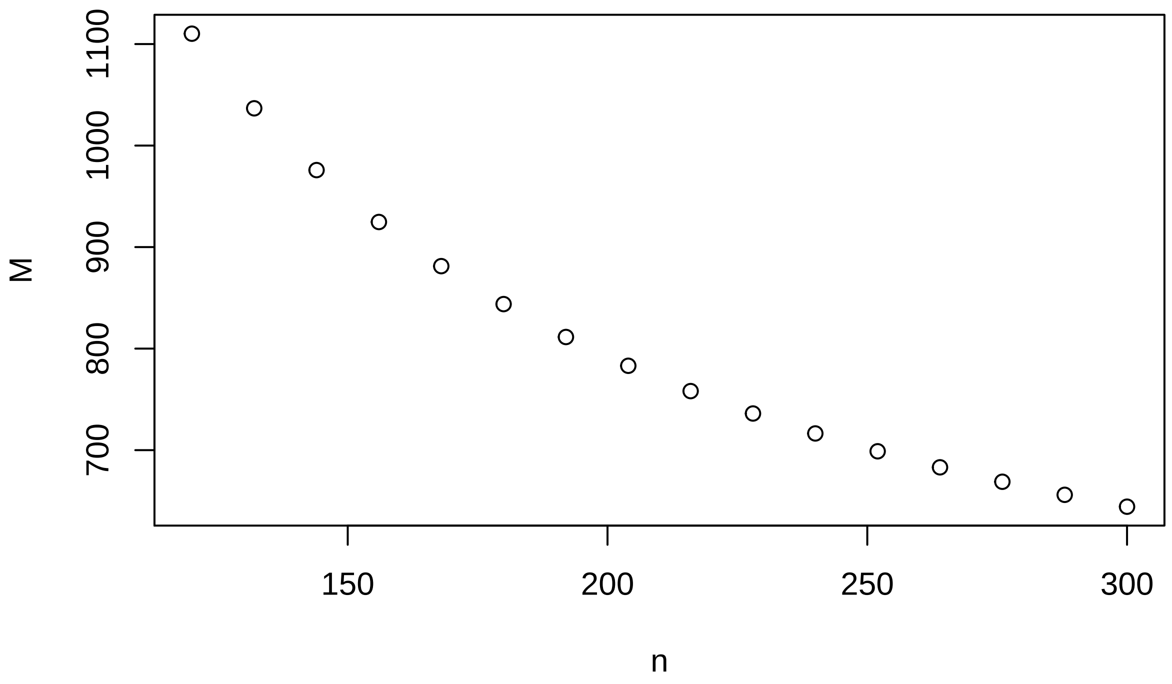
$$n = \frac{\ln\left(\frac{i}{\frac{M}{P}-i}+1\right)}{\ln(1+i)}$$

Lets assume you would can consider repaying between 100 and 1000 euro monthly (in 10 euro steps). In how many months would you have repayed your mortgage? Plot the number of months using `plot(x=M, y=n)`. Explain what happens, if you choose to repay less then 500 euro?

2.6.2 Solutions:

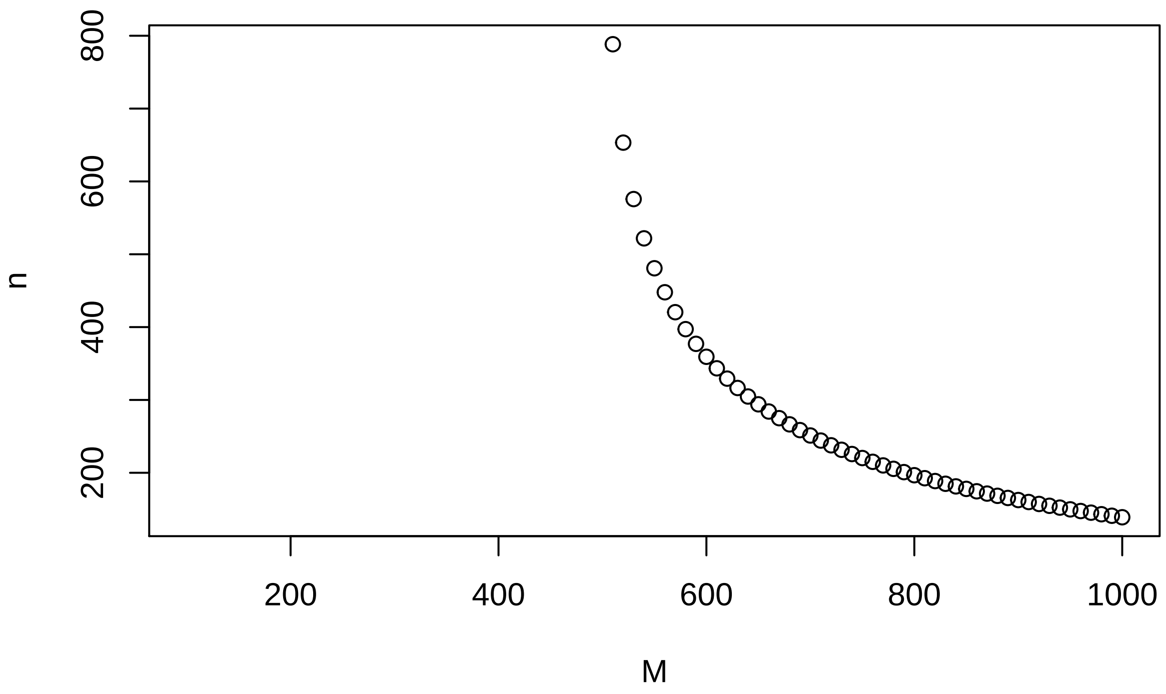
```
P <- 100000
r <- 0.06/12
n <- 15*12
M <- P*(r*(1+r)^n)/((1+r)^n-1)
print(M)
## [1] 843.8568
```

```
P <- 100000
r <- 0.06/12
n <- seq(from=10, to=25)*12
M <- P*(r*(1+r)^n)/((1+r)^n-1)
plot(x=n, y=M)
```



```
P <- 100000
r <- 0.06/12
M <- 650
n <- log(r/(M/P-r)+1)/log(1+r)
years <- n %/% 12
months <- n %% 12
print(years)
## [1] 24
print(months)
## [1] 5.999973
```

```
P <- 100000
r <- 0.06/12
M <- seq(from=100, to=1000, by=10)
n <- log(r/(M/P-r)+1)/log(1+r)
## Warning in log(r/(M/P - r) + 1): NaNs produced
plot(x=M, y=n)
```



2.6.3 Exercise

As a starting data scientist working at Lithuanian Airlines you are tasked to calculate the price for flights between Vilnius and Kaunas, Vilnius and Klaipeda and Vilnius Berlyn. Your boss gave you the following data:

- Vilnius cooridinates: Lat: 54.6870458, Long: 25.2829111
- Klaipeda Lat: 55.7127529, Long:21.1350469
- Berlin: Lat:52.5170365, Long:13.3888599

After googling for "The Great-Circle distance" you find out, that to calculate the distance, you set ϕ_1,λ_1 and ϕ_2, λ_2 to be the geographical latitude and longitude in radians of two points 1 and 2.

The geographical latitude and longitude in radians is calculated as $radians = degrees \times \frac{\pi}{180}$

Then you have to calculate central angle between them using the formula:

$$\Delta\sigma = \arccos\big(\sin\phi_1 \sin\phi_2 + \cos\phi_1 \cos\phi_2 \cos(\Delta\lambda)\big)$$

where $\Delta\lambda$ is tha absolute difference between two longitudes (use `abs()` function). Once the $\Delta\sigma$ is calcultaed, the distance can be calculated as

$$d = r\Delta\sigma$$

with r as earth radius $r = 6371$.

The total cost of each flight is set to be: * fixed cost (airport fee): 50 euro * variable cost: for a selected distance distanceˆ(-0.5) per km of distance

So the break even price for the flight is calculated as

$$BePrice = \text{fixed cost} + distance \times \text{variable cost}$$

```r
# Calculations for Vilnius --> Klaipeda
phi_1 <- 54.6870458
lambda_1 <- 25.2829111
phi_2 <- 52.5170365
lambda_2 <- 21.1350469

phi_1 <- phi_1*pi/180
phi_2 <- phi_2*pi/180
lambda_1 <- lambda_1*pi/180
lambda_2 <- lambda_2*pi/180
```

```
delta_sigma <- acos(sin(phi_1)*sin(phi_2)+cos(phi_1)*cos(phi_2)*cos(abs(lambda_1-lambda_2)))
distance <- delta_sigma*6371

distance
## [1] 364.7552

price_vln_klp <- 50+distance*distance^(-0.5)
print(price_vln_klp)
## [1] 69.09856


# Calculations for Vilnius --> Klaipeda
phi_1 <- 54.6870458
lambda_1 <- 25.2829111
phi_2 <- 55.7127529
lambda_2 <- 13.3888599

phi_1 <- phi_1*pi/180
phi_2 <- phi_2*pi/180
lambda_1 <- lambda_1*pi/180
lambda_2 <- lambda_2*pi/180

delta_sigma <- acos(sin(phi_1)*sin(phi_2)+cos(phi_1)*cos(phi_2)*cos(abs(lambda_1-lambda_2)))
distance <- delta_sigma*6371
distance
## [1] 762.3957

price_vln_bln <- 50+distance*distance^(-0.5)
print(price_vln_bln)
## [1] 77.61151
```

2.7 Vector functions

Usually when applying a function to a vector, the function is applied element by element to the elements of the input vector and returns a result vector with the same length as the input vector.

But some functions are functions that summaryse data. Most common ones are:

min() and max() `mean()`, `median()`, `sd()` and `var()` * `length()`

Many functions take additional arguments, that allow editing the result R retrieves. Check for example `?mean`. Function `mean()` has additional argument `trim` that allows trimming the input vector from both ends. Also the function has an argument `na.rm`, if set to TRUE, all `NA` in the vector will be ommited when appplying the function.

```
# vector
x <- c(-100,1,2,3,4,5,6,7,8,9,1000)

mean(x)
## [1] 85.90909
#triming 10% of elements at the bottom and top of the vector
mean(x, trim = 0.1)
## [1] 5

#vector x with NA
x <- c(-100,1,2,3,NA,5,6,7,8,9,1000)

mean(x)
## [1] NA
# excluding NAs
mean(x, na.rm = TRUE)
## [1] 94.1
# triming 10% of elements at the bottom and top of the vector
mean(x, na.rm = TRUE, trim = 0.1)
## [1] 5.125
```

Some functions, such as `cor()` can take multiple input vectors, e.g. calculating the correlation between two vectors. Check `?cor` for additional arguments the function can take. `cor()` can take different methods: method = c("pearson", "kendall", "spearman")), which the user should select depending on the needs.

```
# loading dataset airquality into R workspace
data("airquality")

# calculating the correlation between Wind and Temp
cor(airquality$Wind, airquality$Temp)
## [1] -0.4579879

# pearson is the standard use
cor(airquality$Wind, airquality$Temp, method = "pearson")
## [1] -0.4579879
# kendall
cor(airquality$Wind, airquality$Temp, method = "kendall")
## [1] -0.3222418
# spearman
cor(airquality$Wind, airquality$Temp, method = "spearman")
## [1] -0.4465408
```

Sometimes you might need to check for `min` or `max` across multiple vectors, but the same positions within the vector, e.g. check if 3 element is greater in vector x or vector y. `max(x,y)` would retrieve one number, the maximum of both vectors x and y. To run `min` and `max` parallel between two or more vectors use `pmin()` and `pmax()`.

```
x <- c(1,9,2,11)
y <- c(2,4,6,8)
```

```r
max(x,y)
## [1] 11
pmax(x,y)
## [1]  2  9  6 11
pmin(x,y)
## [1] 1 4 2 8
```

2.8 Rounding vector elements

There are 5 functions that help user round the elements of the vector or the results of summarizing function.

- `round()`
- `floor()`
- `ceiling()`
- `truncate()`
- `signif()`

Check their all descriptions using `?round`

```
x <- c(-1,-0.99,-.751,-.5,0,0.0013,0.1,0.499,0.5,0.999)
x
##  [1] -1.0000 -0.9900 -0.7510 -0.5000  0.0000  0.0013  0.1000  0.4990  0.5000
## [10]  0.9990
# rounding without arguments
round(x)
##  [1] -1 -1 -1  0  0  0  0  0  0  1
round(x,digits = 1)
##  [1] -1.0 -1.0 -0.8 -0.5  0.0  0.0  0.1  0.5  0.5  1.0
round(x,digits = 2)
##  [1] -1.00 -0.99 -0.75 -0.50  0.00  0.00  0.10  0.50  0.50  1.00

floor(x)
##  [1] -1 -1 -1 -1  0  0  0  0  0  0
ceiling(x)
##  [1] -1  0  0  0  0  1  1  1  1  1
trunc(x)
##  [1] -1  0  0  0  0  0  0  0  0  0
signif(x,digits = 2)
##  [1] -1.0000 -0.9900 -0.7500 -0.5000  0.0000  0.0013  0.1000  0.5000  0.5000
## [10]  1.0000
```

2.9 Cumulative Sums, Products, and Extremes

4 functions return a vector whose elements are the cumulative sums, products, minima or maxima of the elements of the argument. Check the description under `?cumsum`

```
x <- c(1:3,0,10:12)

cumsum(x)
## [1]  1  3  6  6 16 27 39
cummin(x)
## [1] 1 1 1 0 0 0 0
cummax(x)
## [1]  1  2  3  3 10 11 12
cumprod(x)
## [1] 1 2 6 0 0 0 0
```

2.10 Sorting or Ordering Vectors

- `sort()` reorders a vector into ascending or descending order. See `?sort` for more information.
- `order()` order returns a permutation which rearranges its first argument into ascending or descending order, breaking ties by further arguments
- `rank()` returns the sample ranks of the values in a vector. Ties (i.e., equal values) and missing values can be handled in several ways.

```
x <- c(1,3,5,0,2,7,2)
sort(x)
## [1] 0 1 2 2 3 5 7
sort(x, decreasing = TRUE)
## [1] 7 5 3 2 2 1 0

order(x)
## [1] 4 1 5 7 2 3 6
order(x, decreasing = TRUE)
## [1] 6 3 2 5 7 1 4

rank(x)
## [1] 2.0 5.0 6.0 1.0 3.5 7.0 3.5
```

2.10.1 Exercises

1. Load the dataset `AirPassengers` by using the command `data(AirPassengers)`, which contains monthly Airline Passenger Numbers 1949-1960. This dataset is a timeseries object, which will be covered in later lectures. Calculate:

- the mean number of passengers between 1949-1960
- the median number of passengers between 1949-1960
- the minimum number of passengers between 1949-1960
- the maximum number of passengers between 1949-1960
- the standard deviation between 1949-1960

Plot the cumulative sum of passengers between 1949-1960 using `plot(..., type="l)` by substituting ... with the formula needed.

2. What is the sum of highest 2 values in the vector: `c(1001,1000,99999,125,6898,12547,396845,15756)`?

3. What is the sum of lowest 2 values in the vector: `c(1001,1000,99999,125,6898,12547,396845,15756)`?

4. In Lithuania each parsons' income is deducted by a tax-free (NPD) amount, before being taxed with 20 percent income tax. The formula for tax-free (NPD) amount is

$$NPD = 400 - 0.19 * (\text{monthly income} - MW)$$

where `MW` stands for the monthly minimum wage, which equals 607 euro. Calculate the after tax income (net_income) for the following set of income: `c(100,200,300,400,500,600,700,800,900,1000,1500,2000,2500,3000`

Some definitions:

- NPD is the tax free amount $NPD = 400 - 0.19 * (\text{monthly income} - MW)$
- taxable income is $income - NPD$
- labor taxes equal $(income - NPD) \times 0.2$
- net income $net_income = income - (income - NPD) \times 0.2$

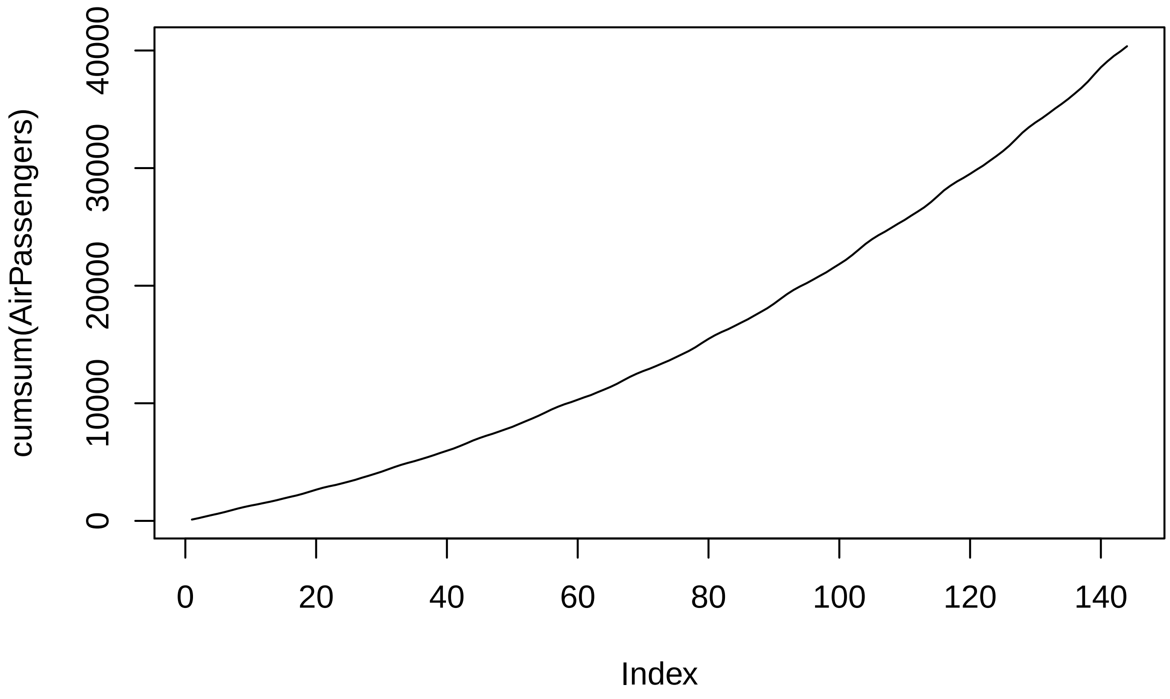
5. Calculate the tax wedge, which is defined as $taxation/income$

2.10.2 Solutions

```
mean(AirPassengers)
## [1] 280.2986
median(AirPassengers)
```

```
## [1] 265.5
min(AirPassengers)
## [1] 104
max(AirPassengers)
## [1] 622
sd(AirPassengers)
## [1] 119.9663

plot(cumsum(AirPassengers), type="l")
```



```
x <- c(1001,1000,99999,125,6898,12547,396845,15756)

# read the second entry of the vector
cumsum(sort(x, decreasing = TRUE))
## [1] 396845 496844 512600 525147 532045 533046 534046 534171

# read the second entry of the vector
cumsum(sort(x))
## [1]    125   1125   2126   9024  21571  37327 137326 534171
```

```
MW <- 607
income <- c(100,200,300,400,500,600,700,800,900,1000,1500,2000,2500,3000,3500,4000)
inc_tax <- 0.2


# after tax income is:
# income - taxable_income *0.2
# where as taxable income is
# income - tax-free income
# and tax free income is
# NPD=400-0.19*(income - MW)
# lets start from the inside
income-MW
##  [1] -507 -407 -307 -207 -107   -7   93  193  293  393  893 1393 1893 2393 2893
## [16] 3393
# for income < MW, values turn negative, lets set them to 0
pmax(income-MW,0)
##  [1]    0    0    0    0    0    0   93  193  293  393  893 1393 1893 2393 2893
## [16] 3393
# calculating tax-free income, values above 2700 turn negative:
400-0.19*pmax(income-MW,0)
##  [1]  400.00  400.00  400.00  400.00  400.00  400.00  382.33  363.33  344.33
## [10]  325.33  230.33  135.33   40.33  -54.67 -149.67 -244.67
# lets set them also to 0
pmax(400-0.19*pmax(income-MW,0),0)
##  [1] 400.00 400.00 400.00 400.00 400.00 400.00 382.33 363.33 344.33 325.33
## [11] 230.33 135.33  40.33   0.00   0.00   0.00
# define this vector as NPD
NPD <- pmax(400-0.19*pmax(income-MW,0),0)
# calculate taxable income
income-NPD
##  [1] -300.00 -200.00 -100.00    0.00  100.00  200.00  317.67  436.67  555.67
## [10]  674.67 1269.67 1864.67 2459.67 3000.00 3500.00 4000.00
# for income less then NPD, the values turn again negative, lets set them to 0
pmax(income-NPD,0)
##  [1]    0.00    0.00    0.00    0.00  100.00  200.00  317.67  436.67  555.67
## [10]  674.67 1269.67 1864.67 2459.67 3000.00 3500.00 4000.00

net_income <- income - pmax(income-NPD,0)* inc_tax


tax_wedge <- pmax(income-NPD,0)* inc_tax /income
tax_wedge
##  [1] 0.00000000 0.00000000 0.00000000 0.00000000 0.04000000 0.06666667
##  [7] 0.09076286 0.10916750 0.12348222 0.13493400 0.16928933 0.18646700
## [13] 0.19677360 0.20000000 0.20000000 0.20000000
```

2.11 Vector logical testing

Logical operators are :

- `<` less than
- `<=` less than or equal to
- `>` greater than
- `>=` greater than or equal to
- `==` exactly equal to
- `!=` not equal to
- `!x` Not x
- `x | y` x OR y
- `x & y` x AND y
- `isTRUE(x)` test if X is TRUE

With R one can perform some logical tests on the elements of a vector (and on elements of other data structers as well)

```
x <- c(-9,-5,-3,0,1,5,7)
x<0
## [1]  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE
x<=0
## [1]  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE
x==0
## [1] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
x>=0
## [1] FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE
x>0
## [1] FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE
x!=0
## [1]  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE
!(x==0)
## [1]  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE
```

```
x <- c(TRUE, FALSE, TRUE, FALSE)
y <- c(TRUE, TRUE, FALSE, FALSE)
a <- c(T,T)
b <- c(F,F)

x&y
## [1]  TRUE FALSE FALSE FALSE
x&&y
## [1] TRUE
x|y
## [1]  TRUE  TRUE  TRUE FALSE
x||y
## [1] TRUE

isTRUE(5>0)
## [1] TRUE
isTRUE(-5>0)
## [1] FALSE
```

2.12 Vector coersion

```
a <- c(0L, 1L, 2L) # same as 0:2
b <- c(0.1, 0.2, 5)
c <- c("alfa", "beta", "gamma")
d <- c(TRUE, FALSE)

class(a)
## [1] "integer"
class(b)
## [1] "numeric"
class(c)
## [1] "character"
class(d)
## [1] "logical"

c(a,b)
## [1] 0.0 1.0 2.0 0.1 0.2 5.0
class(c(a,b))
## [1] "numeric"

c(a,c)
## [1] "0"     "1"     "2"     "alfa"  "beta"  "gamma"
class(c(a,c))
## [1] "character"

c(a,d)
## [1] 0 1 2 1 0
class(c(a,d))
## [1] "integer"

c(c,d)
## [1] "alfa"  "beta"  "gamma" "TRUE"  "FALSE"
class(c(c,d))
## [1] "character"
```

```
a <- 1:5
b <- 1:3
c <- c("yes", "no")

a+b
## Warning in a + b: longer object length is not a multiple of shorter object
## length
## [1] 2 4 6 5 7
a*b
## Warning in a * b: longer object length is not a multiple of shorter object
## length
## [1]  1  4  9  4 10

cbind(a,b)
## Warning in cbind(a, b): number of rows of result is not a multiple of vector
## length (arg 2)
##      a b
## [1,] 1 1
## [2,] 2 2
## [3,] 3 3
## [4,] 4 1
## [5,] 5 2
cbind(a,c)
## Warning in cbind(a, c): number of rows of result is not a multiple of vector
## length (arg 2)
##      a   c
## [1,] "1" "yes"
## [2,] "2" "no"
## [3,] "3" "yes"
## [4,] "4" "no"
## [5,] "5" "yes"
rbind(a,b)
## Warning in rbind(a, b): number of columns of result is not a multiple of vector
## length (arg 2)
##   [,1] [,2] [,3] [,4] [,5]
## a    1    2    3    4    5
## b    1    2    3    1    2
rbind(a,c)
## Warning in rbind(a, c): number of columns of result is not a multiple of vector
## length (arg 2)
##   [,1]  [,2] [,3]  [,4] [,5]
## a "1"   "2"  "3"   "4"  "5"
## c "yes" "no" "yes" "no" "yes"
```

2.13 Subsetting

- `subset`
- `[]`
- `[[]]`
- `$`

```
x <- 1:10
y <- c("alpha", "beta", "gamma")
x[1]
## [1] 1
x[3:6]
## [1] 3 4 5 6
x[c(3,4,5,6)]
## [1] 3 4 5 6
x[c(1,5,9)]
## [1] 1 5 9
x[x<=5]
## [1] 1 2 3 4 5
x[x<2&x>8]
## integer(0)
x[x<2|x>8]
## [1]  1  9 10
y[c(1,3)]
## [1] "alpha" "gamma"
```

2.14 Logical verctors

```r
x <- c(TRUE, TRUE, FALSE)

as.numeric(x)
## [1] 1 1 0

sum(x)
## [1] 2

sum(!x)
## [1] 1
```

2.15 NAs and NANs

```r
x <- c(0,1,NA, 3, 4, NA)
is.na(x)
## [1] FALSE FALSE  TRUE FALSE FALSE  TRUE
sum(is.na(x))
## [1] 2
sum(!is.na(x))
## [1] 4
```