Lecture 2: solutions

Statistical Computing with R

Exercise 2

First, we need to define the matrices:

```
A = matrix(c(-2, 5, 3, 1, -4, 2), nrow = 2, byrow = TRUE)
A
```

```
##      [,1] [,2] [,3]
## [1,]   -2    5    3
## [2,]    1   -4    2
```

To get the proper matrix from a vector (`?c`), we need to specify the number of rows (`nrow`) and/or columns (`ncol`). The argument `byrow` set to TRUE fills the rows of the matrix following the order of the vector, otherwise, the columns are filled following the order of the vector. (`?matrix`)

```
B = matrix(c(-1, 1, 1, -1), nrow = 2, byrow = TRUE)
C = matrix(c(3, 4, 1, -4), nrow = 2, byrow = TRUE)
```

1.

Since both matrices have the same dimensions (2x2) we can directly sum element-wise as follows:

```
B + C
```

```
##      [,1] [,2]
## [1,]    2    5
## [2,]    2   -5
```

2.

To multiply a matrix by a scalar (all the elements of the matrix are multiplied by the same number) we use `*`.

```
B - 2*C
```

```
##      [,1] [,2]
## [1,]   -7   -7
## [2,]   -1    7
```

3.

We use %*% to perform the matrix product of two matrices.

```
B %*% A
```

```
##      [,1] [,2] [,3]
## [1,]    3   -9   -1
## [2,]   -3    9    1
```

This can be performed since the matrices are conformable which means the dimensions are suitable for multiplication. B is (2, 2) and A is (2, 3), so we have the same number of columns in B as the number of rows in A. Notice that the opposite A %*% B cannot be done.

4.

We use the function `t()` to transpose a matrix. In doing so we swap the rows by the columns.

```
cat("Original dimensions:", dim(A), "\n")
```

```
## Original dimensions: 2 3
```

```
cat("Transpose dimensions: ", dim(t(A)))
```

```
## Transpose dimensions:  3 2
```

```
t(A) %*% C
```

```
##      [,1] [,2]
## [1,]   -5  -12
## [2,]   11   36
## [3,]   11    4
```

5.

To calculate the inverse of a matrix we use the function `solve()`. The matrix must be square.

```
solve(C)
```

```
##        [,1]    [,2]
## [1,] 0.2500  0.2500
## [2,] 0.0625 -0.1875
```

Exercise 3

1.

```r
sin(exp(5))
```

```
## [1] -0.6876914
```

2.

We can use the function `sqrt()` or the operator ^ to 0.5 to calculate the square root.

```r
x = c(0, 5, 17) # First we define the vector

sqrt(x + 49)
```

```
## [1] 7.000000 7.348469 8.124038
```

```r
# Or we can compute directly:
(c(0, 5, 17) + 49)^0.5
```

```
## [1] 7.000000 7.348469 8.124038
```

Notice that basic math operations are performed element-wise in the vector. 49 is summed to every value in x. `sqrt()` is applied to every element as well.

3.

```r
round(sqrt(c(0, 5, 17) + 49), 2)
```

```
## [1] 7.00 7.35 8.12
```

4.

Here we need to use ^ to $\frac{1}{3}$

```r
(c(0, 1, 4) + 7)^(1/3)  # Notice the parenthesis
```

```
## [1] 1.912931 2.000000 2.223980
```

5.

`log()` has base = `exp(1)` by default.

```r
log(exp(7) + 12.5^2)
```

```
## [1] 7.133203
```

6.

To specify a different base for the logarithm, we specify the base argument: `base = 4`.

```
log(exp(7) + 12.5^2, base = 4)
```

```
## [1] 5.145518
```

7.

```
x = c(0, 1, 2, 3)
abs(x^2 - 4)
```

```
## [1] 4 3 0 5
```

8.

To perform summation over a vector we simply use the `sum()` function. The easiest way to compute this is to construct a vector of $log_3(\sqrt{x})$ for x = 6,. . . ,20 and sum all its components.

```
x = seq(6, 20) # 6:20 is also valid
sum(log(x^0.5, 3))
```

```
## [1] 17.08889
```

9.

```
x = seq(6, 20)
sum(0.5 * (log(x, 3)))
```

```
## [1] 17.08889
```

Exercise 4

```
# Install and load `remotes`
install.packages('remotes')
library(remotes)

# install `brolgar`
install_github("njtierney/brolgar")
```

```
library(brolgar)
data(heights)
heights = as.data.frame(heights)
data(wages)
wages = as.data.frame(wages)
```

We use the function `as.data.frame()` since the data is in the format of a tibble. A tibble is a type of data frame which is easier to use with large datasets. Since we want to practice using data frames right now we transform the data with the function `as.data.frame()`.

Exercise 5

1.

```
head(heights, 10)
```

```
##        country continent year height_cm
## 1  Afghanistan      Asia 1870    168.40
## 2  Afghanistan      Asia 1880    165.69
## 3  Afghanistan      Asia 1930    166.80
## 4  Afghanistan      Asia 1990    167.10
## 5  Afghanistan      Asia 2000    161.40
## 6      Albania    Europe 1880    170.10
## 7      Albania    Europe 1890    169.80
## 8      Albania    Europe 1900    169.20
## 9      Albania    Europe 2000    167.90
## 10     Algeria    Africa 1910    168.80
```

2.

```
tail(heights, 15)
```

```
##       country continent year height_cm
## 1476    Yemen      Asia 1890   162.720
## 1477    Yemen      Asia 1900   161.990
## 1478    Yemen      Asia 1980   167.600
## 1479    Yemen      Asia 1990   164.400
## 1480   Zambia    Africa 1940   168.063
## 1481   Zambia    Africa 1950   169.635
## 1482   Zambia    Africa 1960   169.719
## 1483   Zambia    Africa 1970   168.932
## 1484   Zambia    Africa 1980   168.194
## 1485 Zimbabwe    Africa 1890   169.045
## 1486 Zimbabwe    Africa 1900   167.630
## 1487 Zimbabwe    Africa 1950   170.988
## 1488 Zimbabwe    Africa 1960   171.076
## 1489 Zimbabwe    Africa 1970   171.340
## 1490 Zimbabwe    Africa 1980   171.004
```

3.

We can use `names()` or `colnames()` functions (`names()` is usually employed for data frames and `colnames()` for matrices). To create a vector with the names, we can assign the output of these functions to a variable.

```r
my_names = names(heights)
# to see the length
length(my_names)
```

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

```r
my_names
```

```
## [1] "country"   "continent" "year"      "height_cm"
```

Exercise 6

1.

```r
nrow(heights)
```

```
## [1] 1490
```

2.

```r
heights[1245, "country"]
```

```
## [1] "Spain"
```

```r
heights[1245, 1]
```

```
## [1] "Spain"
```

Notice that we can get the entries using the names besides the number. This can be very useful sometimes.

3.

Probably the best way to do this is to use the `which()` function. It basically returns which indices are true, see `?which`. To select a variable (column) of the data frame, we can use `$`.

```r
head(heights$country)
```

```
## [1] "Afghanistan" "Afghanistan" "Afghanistan" "Afghanistan" "Afghanistan"
## [6] "Albania"
```

Gives us a vector of names, now we just need to check which entries are equal to “Portugal”.

```r
head(heights$country == "Portugal") # Will return a boolean vector.
```

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

So, using `which()` we have the row number of the observations from Portugal.

```r
which(heights$country == "Portugal")
```

```
##  [1] 1081 1082 1083 1084 1085 1086 1087 1088 1089 1090 1091 1092 1093 1094 1095
## [16] 1096 1097 1098 1099 1100 1101 1102 1103 1104 1105 1106 1107
```

4.

To do this, we can assign the subset of rows to a new variable. We select a vector of rows (since `which` returns a vector with the indices) and all the columns. To select all the columns (or all the rows), it is enough to put a blank space. First rows and second columns.

```r
x = heights[which(heights$country == "Portugal"), ]

# To know how many observations we can use nrow() function:
nrow(x)
```

```
## [1] 27
```

```r
# To know which years we can just select the year column:
x$year
```

```
##  [1] 1720 1730 1740 1750 1760 1770 1780 1790 1800 1810 1820 1830 1840 1850 1860
## [16] 1870 1880 1890 1900 1910 1920 1930 1940 1950 1960 1970 1980
```

5.

To check if the observations have a constantly increasing value, we can either check it directly or use, for example, `diff()` function, see `?diff()`. Our data seems to be already sorted by year so we could use `diff()` directly.

```r
x[, c("year", "height_cm")]
```

```
##      year height_cm
## 1081 1720   163.600
## 1082 1730   164.300
## 1083 1740   164.900
## 1084 1750   165.000
## 1085 1760   165.100
## 1086 1770   164.400
## 1087 1780   163.800
## 1088 1790   163.300
## 1089 1800   164.300
## 1090 1810   165.493
```

```
## 1091 1820   166.338
## 1092 1830   165.292
## 1093 1840   163.539
## 1094 1850   164.160
## 1095 1860   164.066
## 1096 1870   164.688
## 1097 1880   164.161
## 1098 1890   164.150
## 1099 1900   163.774
## 1100 1910   164.491
## 1101 1920   164.880
## 1102 1930   165.570
## 1103 1940   166.385
## 1104 1950   167.410
## 1105 1960   169.240
## 1106 1970   171.380
## 1107 1980   172.130
```

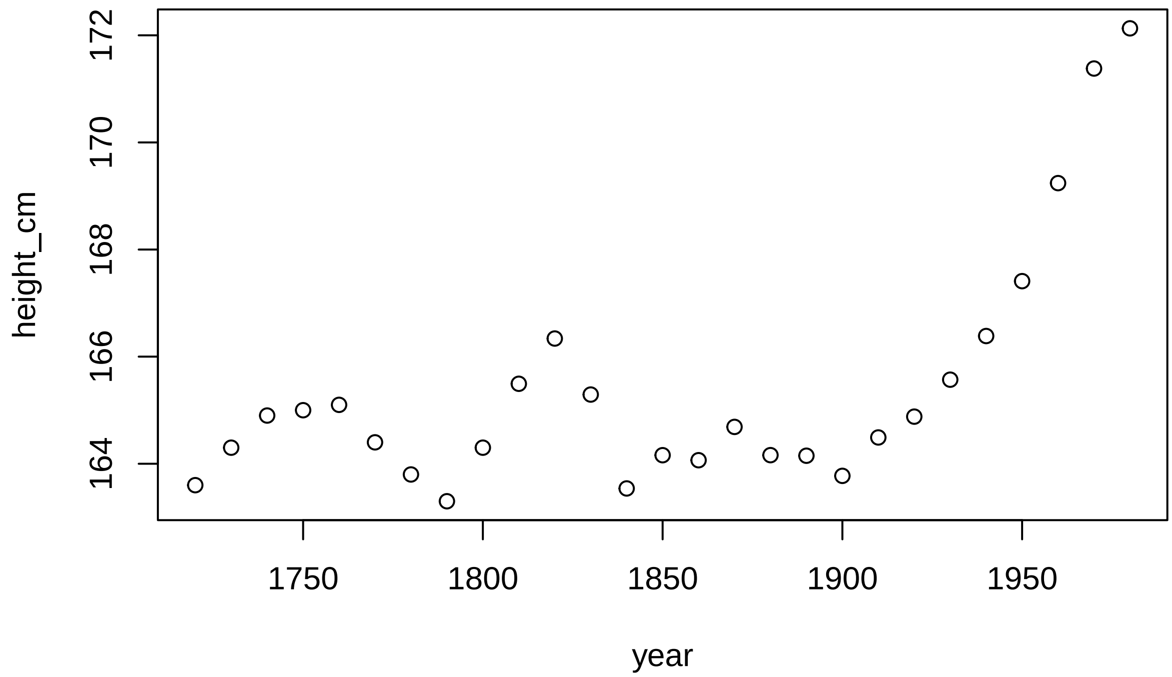
```
# All the rows, just year and height columns.
# We can see that the values are not constantly increasing
```

```
diff(x$height_cm)
```

```
##  [1]  0.700  0.600  0.100  0.100 -0.700 -0.600 -0.500  1.000  1.193  0.845
## [11] -1.046 -1.753  0.621 -0.094  0.622 -0.527 -0.011 -0.376  0.717  0.389
## [21]  0.690  0.815  1.025  1.830  2.140  0.750
```

Not all values are positive. Notice that now there are 26 values and not 27. This is because we use the previous value as a reference.

```
plot(x[, c("year", "height_cm")]) # From the plot is also clear
```



From the plot, we can also see that the mean height in Portugal did not constantly increase over time.

Exercise 7

1.

We can index by a boolean vector as follows:

```
ns = heights[heights$year == 1970, ]
tt = heights[heights$year == 2000, ]
```

2.

We can count the number of entries in the vector to see how many countries are available.

```
length(ns$country)
```

```
## [1] 92
```

```
length(tt$country)
```

```
## [1] 38
```

Notice that the countries are unique. If we had several observations of the same country we could get the different countries using `unique()`.

```
length(unique(ns$country))
```

```
## [1] 92
```

```
length(unique(tt$country))
```

```
## [1] 38
```

3.

```
intersect(ns$country, tt$country)
```

```
##  [1] "Armenia"            "Azerbaijan"         "Bolivia"
##  [4] "Cambodia"           "Colombia"           "Dominican Republic"
##  [7] "Egypt"              "Guatemala"          "Guyana"
## [10] "Honduras"           "Iran"               "Jordan"
## [13] "Laos"               "Morocco"            "Myanmar"
## [16] "Nepal"              "Nicaragua"          "Russia"
## [19] "Singapore"          "Uzbekistan"         "Vietnam"
```

```
#We are selecting the country column.
```

4.

```
tt[tt$country == "Vietnam", "height_cm"]- ns[ns$country == "Vietnam", "height_cm"]
```

```
## [1] 5.2
```

The mean height in Vietnam increased from 1970 to 2000.

Exercise 8

1.

Notice that we use `&` (AND) to get America and 1980. This will create a boolean vector as well.

```
American = heights[heights$continent == "Americas" & heights$year == 1980, ]
Asian = heights[heights$continent == "Asia" & heights$year == 1980, ]
```

In case we needed to match several values for each column we should use `%in%` and not several `&`. See `?"%in%"`.

Another option is to use `subset()`.

```
American = subset(heights, continent == "Americas" & year == 1980)

Asian = subset(heights, continent == "Asia" & year == 1980)
```

2.

We use the `range()` function. It returns minimum and maximum values.

```
cat("Asia:", range(Asian$height_cm), "\n")
```

```
## Asia: 160.1 175.1
```

```
cat("America:", range(American$height_cm))
```

```
## America: 160.8 179.6
```

3.

```
#Asia
median(Asian$height_cm)
```

```
## [1] 165.8
```

```
# America
median(American$height_cm)
```

```
## [1] 170.6
```

4.

We can just use the `max()` function and then get the index of this value. We can also use `which.max()` to perform it directly.

```
American$country[American$height_cm == max(American$height_cm)]
```

```
## [1] "Canada"
```

```
# We index a vector by a boolean vector of the same size.

American$country[which.max(American$height_cm)]
```

```
## [1] "Canada"
```

Exercise 9

1.

```r
?wages
```

We can use the `dim()` function which gives us the rows and columns (in the case of a 2D matrix).

```r
dim(wages)
```

```
## [1] 6402    9
```

2.

We can directly create a new column or variable as follows:

```r
wages$hourly_wage = exp(wages$ln_wages)
```

The number of rows must be equal to the original data frame. Otherwise, it will raise an error.

3.

```r
range(wages$hourly_wage)
```

```
## [1]  2.029927 73.995183
```

4.

```r
mean(wages$hourly_wage)
```

```
## [1] 7.391098
```

```r
median(wages$hourly_wage)
```

```
## [1] 6.309144
```

5.

```r
low_xp = wages[wages$xp <= 2, ]
medium_xp = wages[wages$xp > 2 & wages$xp <= 5 , ]
high_xp = wages[wages$xp > 5, ]
```

6.

```r
mean(low_xp$hourly_wage)
```

```
## [1] 6.241319
```

7.

```r
mean(medium_xp$hourly_wage)
```

```
## [1] 7.135261
```

```r
mean(high_xp$hourly_wage)
```

```
## [1] 8.736166
```

We can observe that the average wage increases with the number of years of work experience.