Question: Advanced R

96

This homework sheet will test your knowledge of data visualization and advanced programming techniques. Note that this sheet requires the use of for and while loops. This is not the most effective way in R to solve most of these tasks but is intended for learning purposes.

Visualization

0

a) Store the following snippet as a CSV file.

File snippet `persons_header.csv`

```
name,height,shoesize,age,gender
Julia,163,39,24,f
Robin,186,44,26,m
Kevin,172,41,21,m
Max,184,43,22,m
Jerry,193,45,31,m
Phillip,175,40,25,m
Christian,184,46,18,m
Marie,178,40,18,f
Sarah,159,37,31,f
Johanna,166,38,24,f
```

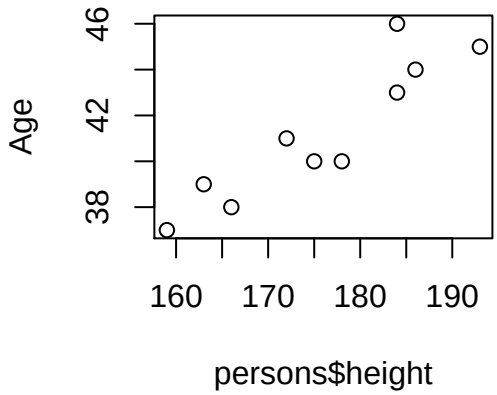
2

b) Load the saved CSV file as variable `persons` and display the height on the x-axis and the shoe size on the y-axis. Name the y-axis accordingly.

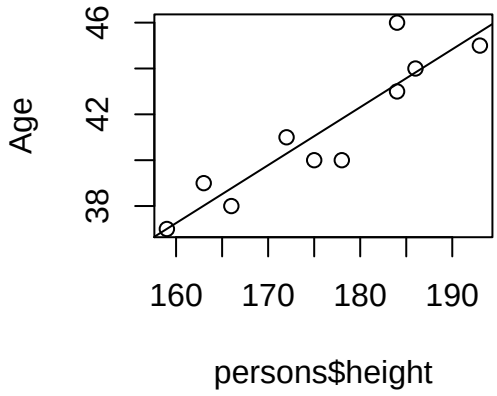
In the next step, create a line of best fit and plot it using the `abline` function.

Solution:

```
plot(persons$height, persons$shoesize,  type='p', ylab="Age")
```

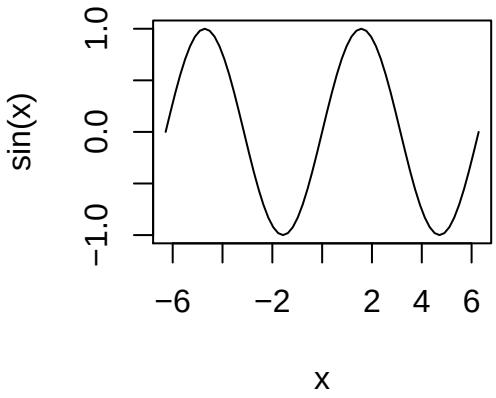



```
plot(persons$height, persons$shoesize,  type='p', ylab="Age")
abline(lm(persons$shoesize ~ persons$height))
```



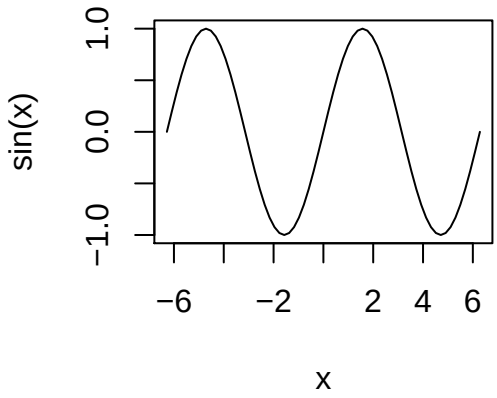
2

c) Recreate the plot given below. Hint: it is a trigonometric function.



Solution:

```r
x <- seq(from = -2*pi,to = 2*pi, by = pi/16)
plot(x,sin(x),type='l')
```



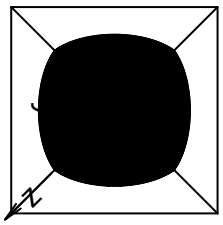
5

d) Create a 3-dimensional plot ranging from -5 to 5 for both x- and y-axis to display the function $f(x, y) = -x^2 - y^2$. View at the plot from straight above.

Solution:

```r
func <- function(x,y) (-x^2 - y^2)

x <- seq(from=-5, to=5, by=0.1)
y <- seq(from=-5, to=5, by=0.1)
z <- outer(x, y, func)
persp(x, y, z, theta=0, phi=90)
```



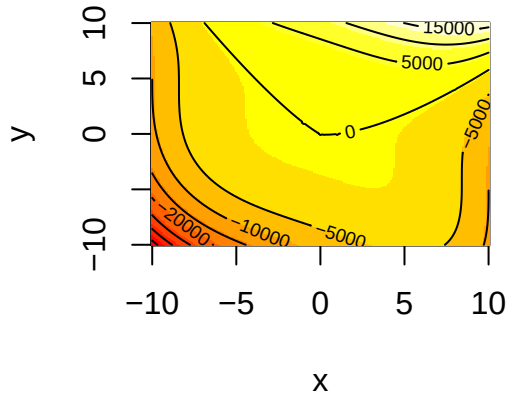
6

e) Create an colored plot with contours of following function $f(x, y) = (x + 2 \cdot y)^3 - x^4$. Both x- and y-dimension should range from -10 to 10.

Solution:

```r
func <- function(x,y) (x+2*y)^3-x^4

x <- seq(-10,10,0.1)
y <- seq(-10,10,0.1)
z <- outer(x,y,func)
image(x,y,z)
contour(x,y,z, add=TRUE)
```



Control Flow

1

f) Define a function that takes name and gender as input and prints the form of address, i. e. `"Johanna"` and `"f"` should print `"Ms. Johanna"`, while `"James"` and `"m"` should result into `"Ms. Johanna"`. Think about what would happen with your function if you enter a `NA` or an empty string `""` as the gender.

Hint: use the function `paste` to concatenate two or more individual strings.

Solution:

```
address <- function(name,gender) {
  if (gender == "f") {
        salutation <- "Ms. "
  } else if (gender == "m") {
    salutation <- "Mr. "
  }
  print(paste(salutation, name, sep=""))
}

address("Sarah", "f")

## [1] "Ms. Sarah"

address("Johanna", "f")
```

```
## [1] "Ms. Johanna"

address("James", "m")

## [1] "Mr. James"

address("Failure", NA) # error

## Error in if (gender == "f") {:  missing value where TRUE/FALSE needed

address("Wrong result", "") # wrong result

## Error in paste(salutation, name, sep = ""):  object 'salutation' not found
```

6

g) Use a for loop and your function created in the previous task to print the form of address for all persons in the `persons_header.csv` file (see above exercise). Do the same using a while loop.

Solution:

```
for (i in 1:nrow(persons)) {
  address(persons$name[i], persons$gender[i])
}

## [1] "Ms. Julia"
## [1] "Mr. Robin"
## [1] "Mr. Kevin"
## [1] "Mr. Max"
## [1] "Mr. Jerry"
## [1] "Mr. Phillip"
## [1] "Mr. Christian"
## [1] "Ms. Marie"
## [1] "Ms. Sarah"
## [1] "Ms. Johanna"
```

```
counter <- 1
while (counter <= nrow(persons)) {
  address(persons$name[counter], persons$gender[counter])
  counter <- counter + 1
}
```

```
## [1] "Ms. Julia"
## [1] "Mr. Robin"
## [1] "Mr. Kevin"
## [1] "Mr. Max"
## [1] "Mr. Jerry"
## [1] "Mr. Phillip"
## [1] "Mr. Christian"
## [1] "Ms. Marie"
## [1] "Ms. Sarah"
## [1] "Ms. Johanna"
```

6

h) Print all **odd** numbers on the screen in the range from 1 to 10 using a for loop. Afterwards, do the same using a while loop.

Hint: you can use the module operator %% to see if a number is divisible by another. Examples are as follows:

```
15 %% 4  # 15 is not divisible by 4

## [1] 3

15 %% 3  # 15 is divisible by 3

## [1] 0

15 %% 5  # 15 is divisible by 5

## [1] 0
```

Solution:

```
for (i in c(1:10)) {
  if(i %% 2 == 1) {
    print(i)
  }
}


## [1] 1
## [1] 3
## [1] 5
## [1] 7
## [1] 9
```

```
i <- 1
while (i <= 10) {
```

```
  print(i)
  i <- i + 2
}

## [1] 1
## [1] 3
## [1] 5
## [1] 7
## [1] 9
```

4

i) Create a vector containing all integer numbers from 7 to 50. Then use a for loop to double every number that is divisible by 7.

Solution:

```
numbers <- c(7:50)
for (i in 1:length(numbers)) {
  if (numbers[i] %% 7 == 0) {
    numbers[i] <- numbers[i] * 2
  }
}

numbers

##  [1] 14  8  9 10 11 12 13 28 15 16 17 18 19 20 42 22 23 24 25 26 27 56 29
## [24] 30 31 32 33 34 70 36 37 38 39 40 41 84 43 44 45 46 47 48 98 50
```

8

j) Write a function that calculates the factorial of a number. Use a non-recursive function to do so. Calculate 10! afterwards. Wha happens if you evaluate your function with input 0, −1, NA or an empty string?

Hint: the factor is defined as $n! = n \cdot (n-1) \cdot \ldots \cdot 2 \cdot 1$. For instance, $5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120$. By definition, $0! := 1$.

Solution:

```
factorial <- function(n) {
  if(n == 0) {
```

```
    return(1)
  }

  fact <- 1
  for (i in 1:n) {
    fact <- fact * i
  }

  return(fact)
}

factorial(5)

## [1] 120

factorial(0)

## [1] 1

factorial(-1) # not defined, i.e. throws wrong result

## [1] 0

factorial(NA) # throws error message

## Error in if (n == 0) {:  missing value where TRUE/FALSE needed
```

2

k) Use the function `factorial` from the previous task to create a function `binom_coeff` that calculates the binomial coefficient for two numbers n and k.

What is the number of possibilities in the German Lotto *6 aus 49* given by $\binom{49}{6}$?

Remember: $\binom{n}{k} = \frac{n!}{k!(n-k)!}$

Solution:

```
binom_coeff <- function(n, k) {
  result <- factorial(n) / (factorial(k) * factorial(n - k))
  return(result)
}

binom_coeff(49, 6) # 13,983,816
```

```
## [1] 13983816
```

3

l) Write a function `fibonacci` that calculates the n-th Fibonacci number. Then calculate the 10-th Fibonacci number.

Definition: $F_1 = 1$, $F_1 = 1$, $F_n = F_{n-1}+F_{n-2}$. this gives the sequence $1, 1, 2, 3, 5, 8, 13, \ldots$

Solution:
Non-recursive approach:

```
fibonacci <- c(1, 1)
for (i in 3:10) {
  fibonacci[i] <- fibonacci[i-1] + fibonacci[i-2]
}
fibonacci[10]
```

```
## [1] 55
```

Recursive approach:

```
fibonacci <- function(n) {
  if (n == 2 || n == 1) {
    return(1)
  }
  return (fibonacci(n-2) + fibonacci(n-1))
}

fibonacci(10)
```

```
## [1] 55
```

2

m) Write a function `digit_sum` that calculates the digit sum of a given input number. A mathematical solution or a string solution can be used. Then calculate the digit sum of 942.

Example: the digits sum of 17311 is $1 + 7 + 3 + 1 + 1 = 13$.

Solution:

```r
# using string manipulation
digit_sum <- function(number) {
  digits <- as.numeric(unlist(strsplit(as.character(number), split="")))
  return(sum(digits))
}
digit_sum(942)

## [1] 15

# using maths
digit_sum <- function(number) {
  sum <- 0
  while (number != 0) {
    sum <- sum + (number %% 10)
        number <- floor(number / 10)
  }

  return(sum)
}

digit_sum(942)

## [1] 15
```

7

n) Create a function `sum_to_n` that takes a positive number n as input and returns the sum from 1 to n using (A) vector operations and (B) a for loop. Then create a third variant (C) that takes the same input and returns the sum from 1 to n using the numeric formula. Test all three variants yield the same results for $n = 10$

Numeric formula: $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

Solution:
Variant (A)

```r
sum_to_n <- function(n) {
  return(sum(1:n))
}
sum_to_n(10)

## [1] 55
```

Variant (B)

```
sum_to_n <- function(n) {
  sum <- 0
  for (i in 1:n) {
    sum <- sum + i
  }
  return(sum)
}
sum_to_n(10)

## [1] 55
```

Variant (C)

```
sum_to_n <- function(n) {
  return((n*(n+1))/2)
}
sum_to_n(10)

## [1] 55
```

Vector Indexing

5

o) Copy the R code below. Write a for loop to find the index of first `"apple"` element in the vector and print it. Then use another for loop to find all `"apple"` elements in the vector and print their indices.

```
fruit_basket <- c("pineapple", "apple", "pear", "orange", "apple",
                  "pomegranate", "apple", "apple")
```

Hint: look up the functionality of the keyword `break`

Solution:
Find the first index:

```
for (i in 1:length(fruit_basket)) {
  if (fruit_basket[i] == "apple") {
    print(i)
    break
  }
}

## [1] 2
```

Find all indices:

```
for (i in 1:length(fruit_basket)) {
  if (fruit_basket[i] == "apple") {
    print(i)
  }
}

## [1] 2
## [1] 5
## [1] 7
## [1] 8
```

Nested Loops

8

p) Copy the R code below. Use two nested for loops to check the array `essay` for misspelling of words using the `dictionary`. Print misspelled words.

```
dictionary <- c("brown", "dog", "fox", "jumps",
                "lazy", "over", "quick", "the")
essay <- c("the", "quik", "brown", "fox",
           "jumpps", "over", "the", "lazy","dok")
```

Solution:

```
for (word in essay) {
  error <- TRUE
  for (correct_spelling in dictionary) {
    if(word == correct_spelling) {
          error <- FALSE
        }
  }
  if (error) {
    print(word)
  }
}

## [1] "quik"
## [1] "jumpps"
## [1] "dok"
```

7

q) Copy the R code below. Write a function `find` that takes a matrix and an element as arguments. The function should return the index of the first occurrence of the element if the matrix contains the element. Use two for loops to realize this function. The matrix should be searched row by row.

```
mat <- matrix(c(1,7,3,3,5,1,0,12,4), nrow=3, ncol=3)
```

Solution:

```
find <- function(mat, element) {
  for (row in 1:nrow(mat)) {
    for (col in 1:ncol(mat)) {
      if (mat[row, col] == element) {
            return(c(row, col))
          }
    }
  }
  return(-1)
}

find(mat, 12)

## [1] 2 3
```

12

r) Print all **prime** numbers on the screen in the range from 1 to 20 using a for loop. Afterwards, do the same using a while loop. Think about reasons why these algorithms might not be efficient.

Hint: primes are all those numbers above or equal to 2 that have only 1 or themselves as a divisor.

Solution:

```
for (i in c(2:20)) {
  isPrime <- TRUE
  for (j in 2:(i-1)) {
    if (i %% j == 0) {
      isPrime <- FALSE
    }
  }
```

```
  if (isPrime || i == 2) {
    print(i)
  }
}
```

```
## [1] 2
## [1] 3
## [1] 5
## [1] 7
## [1] 11
## [1] 13
## [1] 17
## [1] 19
```

```
i <- 2
while (i <= 20) {
  isPrime <- TRUE
  j <- 2
  while (j <= i-1) {
    if(i %% j == 0) {
      isPrime <- FALSE
    }
    j <- j + 1
  }
  if (isPrime) {
    print(i)
  }
  i <- i + 1
}
```

```
## [1] 2
## [1] 3
## [1] 5
## [1] 7
## [1] 11
## [1] 13
## [1] 17
## [1] 19
```

Apparently, the code checks all possible divisors from 1 to n. However, it would be sufficient to test only the range up to $\sqrt{n}$. In addition, the inner loops are executed even a divisior has already been found.

6

s) Write a function `gcd` that takes two numbers as input. The function should return the greatest common divisor of those two numbers. Use the euclidean algorithm to

find it.

What is the greatest common divisor of 1071 and 462?

Hint: you can find details on the Euclidean algorithm in the Wikipedia, see `https://en.wikipedia.org/wiki/Euclidean_algorithm`

Solution:
Recursive approach:

```
gcd <- function(a, b) {
  if (b == 0) {
    return(a)
  }

  return(gcd(b, a %% b))
}

gcd(1071, 462)

## [1] 21
```

Non-recursive approach:

```
gcd <- function(a, b) {
  while(a != b) {
    if (a > b) {
          a <- a - b
        } else {
          b <- b - a
        }
  }
  return(a)
}

gcd(1071, 462)

## [1] 21
```

Timing

2

t) Determine whether squaring or calculating the square root is faster in R. Use a vector with inputs from 1 to $10,000,000$.

Solution:

```
# squares are faster
system.time((1:10000000)^2)

##    user  system elapsed
##    0.12    0.02    0.16

system.time(sqrt(1:10000000))

##    user  system elapsed
##    0.13    0.02    0.16
```

Advanced Plotting

2

u) Use the library `ggplot2` to plot heights (x-axis) and corresponding shoe sizes (y-axis) of the persons in `persons_header.csv`. Use a different symbol to clarify which gender the person belongs to and color them to show their ages.

Hint: Checkout the slides on *ggplot2: An implemenation of the grammar of graphics* by Hadley Wickham at `http://ggplot2.org/resources/2007-vanderbilt.pdf`

Solution:

```
library(ggplot2)
ggplot(persons,
       aes(x=persons$height, y=persons$shoesize, shape=persons$gender,
           color=persons$age)) +
  geom_point(size=10)
```