SRT411A0

Satyajit Nandi

February 20, 2018

ToDo (3.1)

Compute the difference between 2018 and the year you started at this university and divide this by the difference between 2014 and the year you were born. Multiply this with 100 to get the percentage of your life you have spent at this university. Use brackets if you need them.

`Calculating percent spent at the university`

$$PercentofLifeinCollege = (CollegeYears/Age) * 100$$

```
((2018-2017)/(2018-1983))*100
```

```
## [1] 2.857143
```

ToDo (3.2)

Repeat the previous ToDo(3.1), but with several stps in between. You can give the variables any name you want, but the name has to start with letter.

```
PresentTime = 2018
Born = 1983
SchoolStart = 2017

((PresentTime - SchoolStart)/(PresentTime - Born))*100
```

```
## [1] 2.857143
```

ToDo (3.4)

Compute the sum of 4,5,8 and 11 by first combining them into a vector and then using the function sum.

```
4+5+8+11
```

```
## [1] 28
```

Using function we can computer as follows:

```
a=c(4,5,8,11)
sum(a)
```

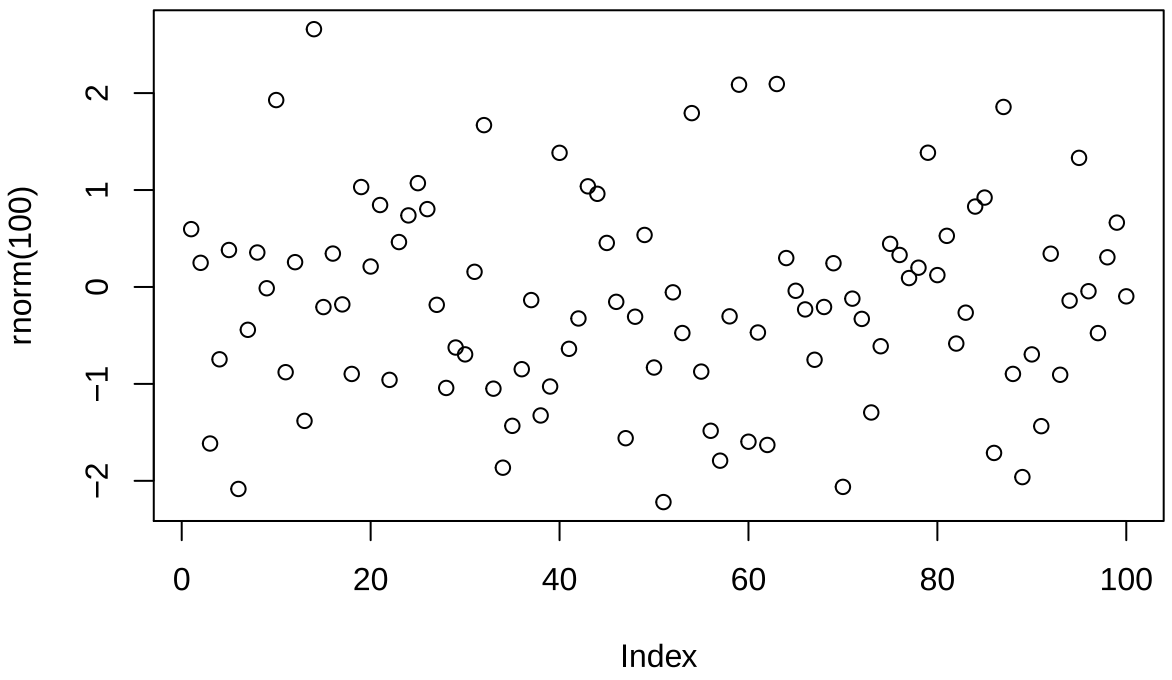
```
## [1] 28
```

Note that, here c is a function which is short for concatenate.

ToDo (3.5)

Plot 100 normal random numbers

```
plot(rnorm(100))
```



ToDo (4)

Find help for the sqrt function

```
help("sqrt")
```

or another way to get help:

```
?sqrt
```

ToDo (5)

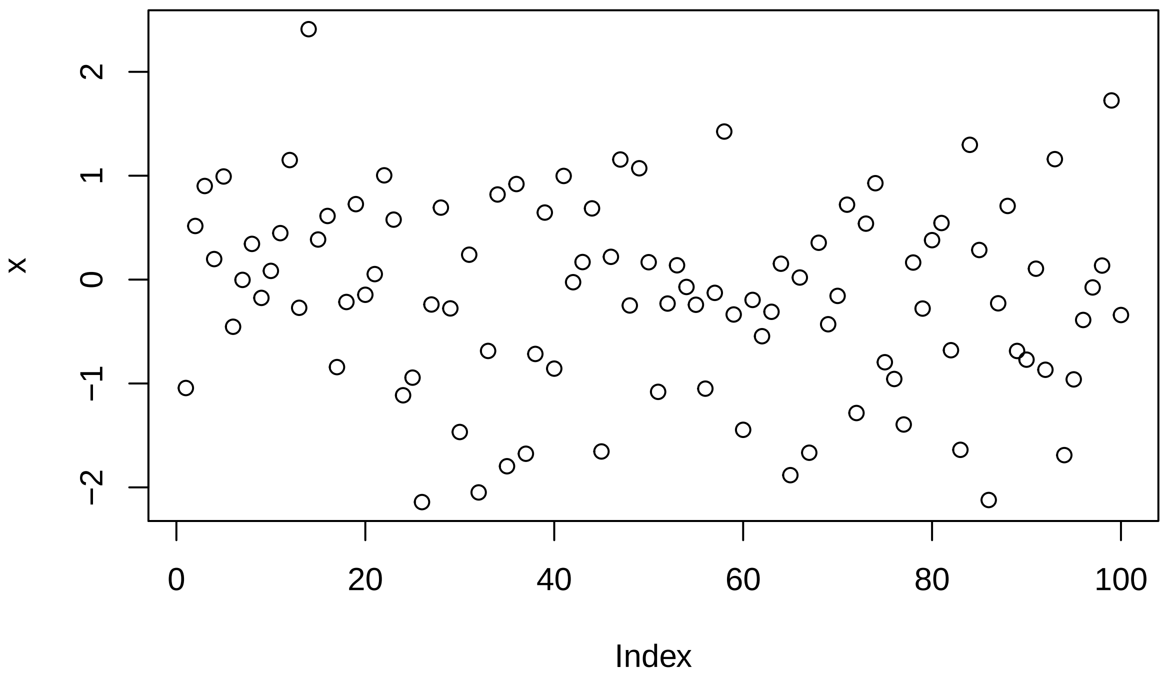
Make a file called *firstscript.R* containing R-code that generates 100 random numbers and plots them, and run this script several times.

First to save R script names as *firstscript.R* into the R working directory.

`plot(rnorm(100))`

To run this script in R:

```
source("firstscript.R")
```



To run this script multiple times we can press **Ctrl+Alt+s**.

ToDo (6.2)

Put the numbers 31 to 60 in a vector named P and in a matrix with 6 rows and 5 columns named Q.

```
P = seq(from=31, to=60)
#Printing result P
P
```

```
##  [1] 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53
## [24] 54 55 56 57 58 59 60
```

```
Q = matrix(data = P, nrow = 6, ncol = 5)
#printing result Q
Q
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   31   37   43   49   55
## [2,]   32   38   44   50   56
## [3,]   33   39   45   51   57
## [4,]   34   40   46   52   58
## [5,]   35   41   47   53   59
## [6,]   36   42   48   54   60
```

ToDo (6.3) & (7)

Make a script file which constructs three random normal vectors of length 100. Call these vectors x1, x2, and x3. Make a data frame called t with three colums (called a, b, and c) containing respectively x1, x1+x2 and x1+x2+x3. Call plot(t) for this data frame.

```
# constructs three random normal vectors of lenth 100
x1=rnorm(100)
x2=rnorm(100)
x3=rnorm(100)
```

```r
# Call these variables
x1
```

```
##   [1]  2.24837848  0.35587210  0.41824859 -2.06701364 -0.93554359
##   [6] -0.43976996  0.55960028  1.42332082  1.35482190 -1.28127792
##  [11] -0.16133367 -0.43475157  0.44007317 -0.59938242  0.62506435
##  [16]  0.58391087  1.11095338 -1.07414538  0.59494644  0.36948551
##  [21] -0.50422051 -0.51160002  1.52518189  1.37843526  0.66379396
##  [26] -1.26485970 -0.77339700 -0.81559320 -0.94057375  1.02798477
##  [31]  0.49642483 -2.22543295  0.05775596 -2.37681428  0.20361496
##  [36]  0.89892748 -0.49494326 -0.37458333  2.88603019 -1.60009803
##  [41] -0.07620271 -0.08364633 -0.61513573 -0.03710623 -0.75583071
##  [46] -1.82903577  1.52368898  2.10270904 -0.98393419 -0.61224500
##  [51]  0.18682723  1.35533110  1.71293519  0.53388211 -0.77602383
##  [56]  0.40785606  0.09022485 -0.17161945 -1.16968467 -0.14990954
##  [61]  0.25863020 -0.95925400 -0.95120513  1.17116906  1.47964032
##  [66]  0.58716103 -0.44074529  2.19692363 -0.65987828  0.25653040
##  [71]  1.77744918 -0.81629439  0.10070301 -0.67073400  1.17054255
##  [76]  0.21053671 -0.81635075  0.45522697  0.80306899 -0.49176847
##  [81] -0.39835304 -0.46033956 -1.26511239 -0.65973027 -0.09777823
##  [86] -0.04059963  0.61533862 -1.56845709 -0.69359495  0.27799599
##  [91]  1.93133338 -0.49262333 -0.71744626  1.95006549  0.55829809
##  [96]  1.95438771  0.62597642  0.18963517 -1.78892048 -1.33988686
```

```r
x2
```

```
##   [1] -0.37602059  0.41162790  0.34600597  0.43499854 -0.15023395
##   [6] -0.90877250  0.85872911 -0.89167498 -0.71162959  1.66956739
##  [11] -1.25666979 -2.13265815  1.58748621  0.95448920 -1.04296291
##  [16]  0.55554490  1.70639157 -0.34567584 -0.11687608 -1.69201830
##  [21] -0.87447861 -0.56888001 -0.74448533 -1.24262149  0.05691976
##  [26] -1.34281466 -1.75092663 -1.84204617  0.86386979 -0.53282256
##  [31] -0.90536420  1.04446938  0.53457468  0.57512547 -0.33179661
##  [36]  1.84726585  0.26880911  1.24765145  0.42688681 -1.11139296
##  [41] -0.38278581 -0.89711524 -0.26236362 -1.84077954 -0.43723138
##  [46]  0.75010156  0.53953050  0.45987389  0.52475951 -1.16905896
##  [51]  0.71503465  0.13344839 -0.84775558  0.90596668  0.09776601
##  [56]  0.71846637  1.29533037 -0.14608820 -0.57118463 -0.26750679
##  [61]  0.49421364  1.62497680  0.49211830  1.04151932 -0.92247648
##  [66]  0.16838428 -1.46119643 -0.97362982 -1.11516428 -1.31316829
##  [71]  0.51949109  1.10509660 -0.37818249  0.36051519 -0.57470248
##  [76]  0.75835777 -0.43877252 -0.27708791 -1.43782334  0.60664601
##  [81]  1.10453255 -1.88296487 -0.92554761  0.70163141 -0.14326690
##  [86] -0.23830737 -0.22803414 -1.19783369  0.99534273 -0.03614725
##  [91]  0.06449590 -0.35183575  1.76919949  0.40319734 -1.87675777
##  [96]  0.33052124  1.39557829 -1.24480074  2.28564749  3.06345198
```

```r
x3
```

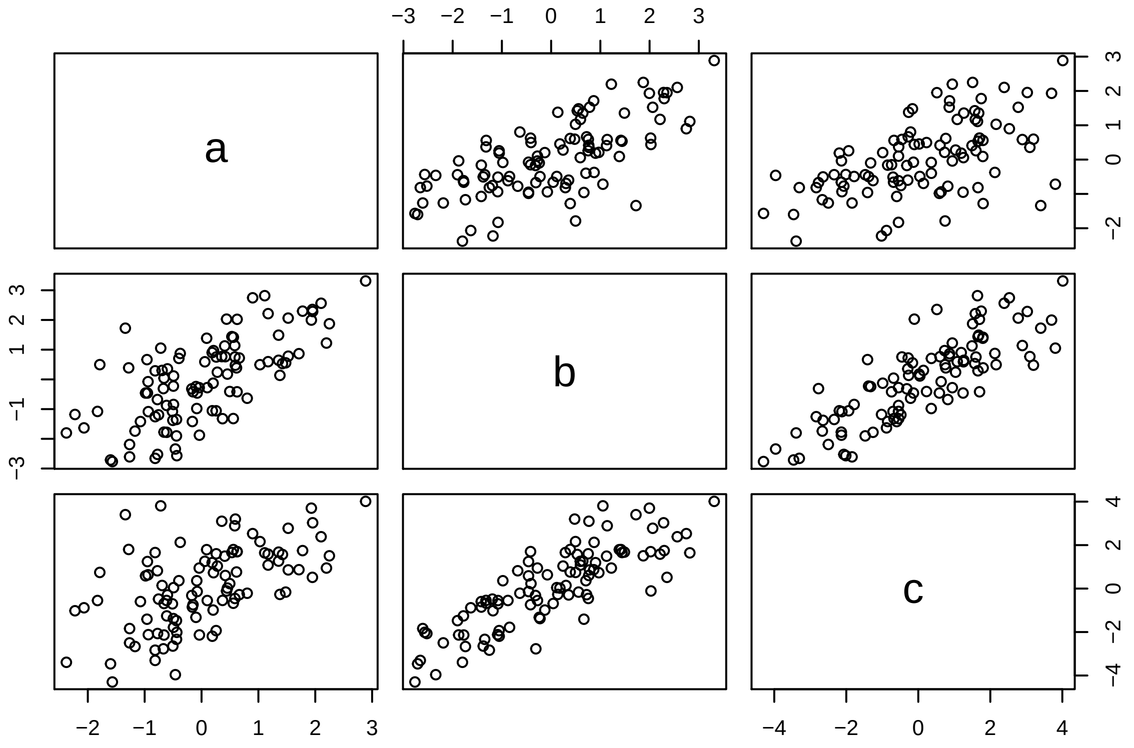
```
##   [1] -0.3637602668  2.3311573232 -0.1612232020  0.7487786233 -1.0350556614
##   [6] -0.9876439771  0.3768715221  1.0365803780  0.6209324574  1.4120839725
##  [11]  0.5674313650  0.5542433901 -2.1373348637 -0.6474838599  2.1196547563
##  [16]  1.7506194341 -1.1733413980  0.8210285526  2.7198824109  0.7782125624
##  [21] -1.2664151684  0.3784330639  0.0783849494 -0.4044649929 -1.0004324718
##  [26]  0.7681296889  0.4565445122 -0.6474875809  0.7106696221  1.6665463871
##  [31]  0.6377339357  0.1600441629  0.6617967359 -1.5903609833 -0.8600661240
##  [36] -0.2175576827 -1.1540839124  1.2535279827  0.6998168947 -0.7514539592
##  [41]  0.3238516360  1.3414657440  0.3296702197 -0.2560725182  0.7094147955
##  [46]  0.5292742621  0.7120037045 -0.1772281787  1.0447503912  0.5235515427
##  [51]  0.2907288786  0.1890332976  0.0035544081  0.2195193218  1.4979426962
##  [56]  0.3653160378  0.4061569418 -0.0002335639 -0.9246350756 -0.3246914845
##  [61]  0.8487145064 -2.0750283281  1.7028369011 -0.6286049462 -0.7191489918
##  [66] -1.2100864392  0.4272474267 -0.2782554619 -0.3616020019 -0.8773501788
##  [71] -0.5494693673  1.3686010458 -0.2719053567 -2.4611473531  0.4859132547
##  [76] -0.2379989647 -1.5791110095 -0.1543353378  0.4202242048 -0.0712679095
##  [81] -0.3433214224 -1.6182318192 -0.3076783388 -0.7290088850 -1.0822312609
##  [86]  1.2239870657  0.3763375547 -1.5320982640 -0.1592413851  0.7934164092
##  [91]  1.7054717091 -0.9372615877  2.7549661784 -1.8367923926  0.6430822073
##  [96]  0.7418660384 -0.3236764207 -1.1371196359  0.2467236712  1.6771712049
```

```
# Make a data frame with three columns

t = data.frame(a = x1, b = (x1+x2), c = (x1+x2+x3))

# Call plot(t)

plot(t)
```



```
# call sd(t)

sd(t$a)
```

```
## [1] 1.081518
```

```
sd(t$b)
```

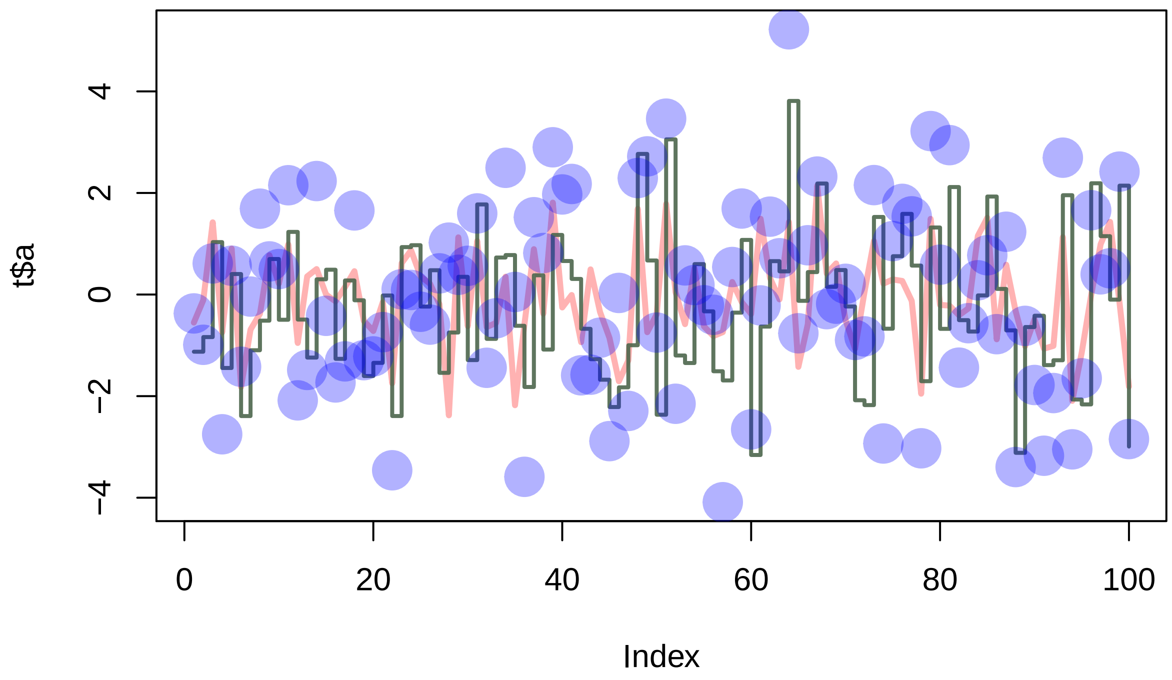
```
## [1] 1.428054
```

```
sd(t$c)
```

```
## [1] 1.861209
```

ToDo (7)

Make graphics of the previous data



```
**rgb** function creats colors corresponding to the given intensities of the red, green and blue primati
Usage: rgb(red, green, blue, alpha, names = NULL, maxColorValue=1)

**lwd** is the line width, a positive nummber, defaulting to 1.

**pch** is plotting 'character', i.e., symbol to use. Either an integer specifying a symbol or a single

**cex** A numerical value giving the amount by which plotting text and symbols should be magnified rela
```

ToDo (8)

```
# setting up data frame
d = data.frame(a=c(1,2,3,4,16,32), g=c(2,3,4,16,32,64), x=c(3,6,12,24,48,96))

# writing into file names tst1.txt
write.table(d, file="tst1.txt", row.names = FALSE)

#reading from file tst1.txt
d2=read.table(file="tst1.txt", header = TRUE)

# printing d2
d2
```

```
##    a  g  x
## 1  1  2  3
## 2  2  3  6
## 3  3  4 12
## 4  4 16 24
## 5 16 32 48
## 6 32 64 96
```

```r
#multiplying column g by 5 and write into a file named tst2.txt
write.table(d2$g*5, file = "tst2.txt", row.names = FALSE, col.names = "g")
```

ToDo (9)

Compute the mean of the square root of a vector of 100 random numbers. What happens?

```r
mean(sqrt(rnorm(100)))
```

```
## Warning in sqrt(rnorm(100)): NaNs produced
```

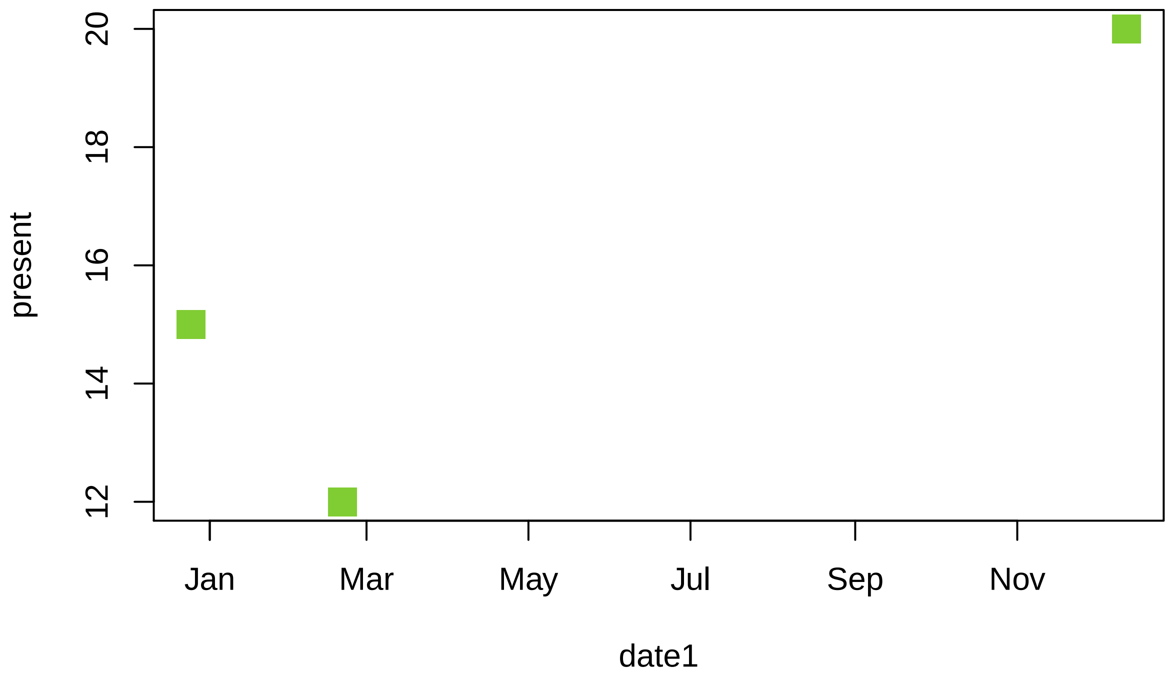
```
## [1] NaN
```

This will not work. This is because rnorm produces numbers that doesn't have a proper square root (NaNs). The square root of negative number is only possible in complex number.

ToDo (10.2)

Make a graph with on the x-axis: today, Sinterklass 2017 and your next birthday and on the y-axis the number of presentes you expect on each of these days.

```r
date1=strptime(c("20180220", "20171225", "20181212"), format="%Y%m%d")
present=c(12, 15, 20)
plot(date1,present, type="p", pch=15, cex=2, col=rgb(0.5,.8,.2,1))
```



ToDo (11.2)

Make a vector from 1 to 100. Make a for-loop which runs through the whole vector. Multiply the elements which are smallar than 5 and larger than 90 with 10 and the other elements with 0.1

```
# defining variable from 1 to 100 and empty result variable
var1 = seq(from=1, to=100)
result = c()

# for-loop
for (i in var1)
    {
     if (i<5 | i>90) # elements are smallar than5 and larger than 90
       {result[i] = var1[i] * 10}
     else { result[i] =var1[i] * 0.1}
    }
# viewing result
result
```

```
##   [1]   10.0   20.0   30.0   40.0    0.5    0.6    0.7    0.8    0.9    1.0
##  [11]    1.1    1.2    1.3    1.4    1.5    1.6    1.7    1.8    1.9    2.0
##  [21]    2.1    2.2    2.3    2.4    2.5    2.6    2.7    2.8    2.9    3.0
##  [31]    3.1    3.2    3.3    3.4    3.5    3.6    3.7    3.8    3.9    4.0
##  [41]    4.1    4.2    4.3    4.4    4.5    4.6    4.7    4.8    4.9    5.0
##  [51]    5.1    5.2    5.3    5.4    5.5    5.6    5.7    5.8    5.9    6.0
##  [61]    6.1    6.2    6.3    6.4    6.5    6.6    6.7    6.8    6.9    7.0
##  [71]    7.1    7.2    7.3    7.4    7.5    7.6    7.7    7.8    7.9    8.0
##  [81]    8.1    8.2    8.3    8.4    8.5    8.6    8.7    8.8    8.9    9.0
##  [91]  910.0  920.0  930.0  940.0  950.0  960.0  970.0  980.0  990.0 1000.0
```

ToDo (11.3)

Write a function for the previous ToDo, so that you can feed it any vector you like (as argument). Use a for-loop in the function to do the computation with each element. Use the standard R function length in the specification of the counter.

```
myfunc = function(arg1)
{
    b = length(arg1) #defining length of the argument
    for (i in 1:b)
    {
        if(arg1[i]<5 | arg1[i]>90) #creating condition
            {
                arg1[i]=arg1[i]*10
            }
        else
            {
                arg1[i]=arg1[i]*0.1
            }
    }
    return(arg1)
}

#calling the function
myfunc(arg1=1:100) # or, myfunc(1:100)
```

```
##   [1]   10.0   20.0   30.0   40.0    0.5    0.6    0.7    0.8    0.9    1.0
##  [11]    1.1    1.2    1.3    1.4    1.5    1.6    1.7    1.8    1.9    2.0
##  [21]    2.1    2.2    2.3    2.4    2.5    2.6    2.7    2.8    2.9    3.0
##  [31]    3.1    3.2    3.3    3.4    3.5    3.6    3.7    3.8    3.9    4.0
## [41]    4.1    4.2    4.3    4.4    4.5    4.6    4.7    4.8    4.9    5.0
## [51]    5.1    5.2    5.3    5.4    5.5    5.6    5.7    5.8    5.9    6.0
## [61]    6.1    6.2    6.3    6.4    6.5    6.6    6.7    6.8    6.9    7.0
## [71]    7.1    7.2    7.3    7.4    7.5    7.6    7.7    7.8    7.9    8.0
## [81]    8.1    8.2    8.3    8.4    8.5    8.6    8.7    8.8    8.9    9.0
## [91]  910.0  920.0  930.0  940.0  950.0  960.0  970.0  980.0  990.0 1000.0
```

```
#alternate way to making the same function

myfunc=function(arg1)
{
  ifelse(arg1<5 | arg1>5, arg1*10, arg1*0.1)
}

myfunc(1:100)
```

```
##   [1]   10.0   20.0   30.0   40.0    0.5   60.0   70.0   80.0   90.0  100.0
##  [11]  110.0  120.0  130.0  140.0  150.0  160.0  170.0  180.0  190.0  200.0
##  [21]  210.0  220.0  230.0  240.0  250.0  260.0  270.0  280.0  290.0  300.0
##  [31]  310.0  320.0  330.0  340.0  350.0  360.0  370.0  380.0  390.0  400.0
##  [41]  410.0  420.0  430.0  440.0  450.0  460.0  470.0  480.0  490.0  500.0
##  [51]  510.0  520.0  530.0  540.0  550.0  560.0  570.0  580.0  590.0  600.0
##  [61]  610.0  620.0  630.0  640.0  650.0  660.0  670.0  680.0  690.0  700.0
##  [71]  710.0  720.0  730.0  740.0  750.0  760.0  770.0  780.0  790.0  800.0
##  [81]  810.0  820.0  830.0  840.0  850.0  860.0  870.0  880.0  890.0  900.0
##  [91]  910.0  920.0  930.0  940.0  950.0  960.0  970.0  980.0  990.0 1000.0
```