Stat 243
Problem set 5

A. Strand, ID: 540441, GitHub: AndreasStrand

October 16, 2015

Problem 1

(a) If we store 1.000000000001 on a computer we will have 13 digits of accuracy.

(b) The sum(x) in R gives the right mathematical result with the accuracy expected from (a). The R code for problem 1 is included as code 1.

Code 1: Summing a big number and small numbers in R using different approaches.

```
# Sum of 1 + 1e-16 +... + 1e-16 with sum()
x = c(1, rep(1e-16, 1e4))
xSum = sum(x) # 1.000000000001

# Sum of 1 + 1e-16 +... + 1e-16 as a for loop
bigFirst = 0
for(i in 1:length(x)) {
  bigFirst = bigFirst + x[i]
}
sprintf("%.20f",bigFirst) # 1.00....0

# Sum of 1e-16 +... + 1e-16 + 1 as a for loop
smallFirst = 0
for(i in length(x):1) {
  smallFirst = smallFirst + x[i]
}
smallFirst = smallFirst + 1
sprintf("%.20f",smallFirst) # 1.00000000000100008890
```

(c) The sum function in R does not give the right mathematical result, but returns 1 instead. There is by this a difference in how the sum() function works in python versus r. The python code for problem 1 is included in code 2

Code 2: Summing a big number and small numbers in Python using different approaches.

```
# Importing packages for creating an array and displaying
# decimal numbers with high precision
import numpy as np
from decimal import *

# Creating the vector x
x = np.array([1e-16]*(10001))
x[0] = 1

# Summing over x using the sum() function
xSum = sum(x)
print(Decimal(xSum)) # 1

# Summing over x forward using a for loop
bigFirst = 0
for i in x:
    bigFirst = bigFirst + i
```

```
print(Decimal(bigFirst)) # 1

# Summing over x backwars using a for loop
smallFirst = 0
for i in reversed(x):
    smallFirst = smallFirst + i
print(Decimal(smallFirst)) # 1.0000000000100008890...
```

(d) The answer of the sum of x using a *for* loop from the first entry is 1. When adding the 1 in the end the answer is 1.000000000001, which is mathematically correct. The answer has 16 decimal places of accuracy as expected. The results and accuracy are the same in Python as in R when using a *for* loop.

(e) The answers from (*b*) and (*d*) suggest that R's sum() function is not simply summing from left to right, because the result then would be 1 instead of the mathematical result. It works in a smarter way. A theory is that it first sort the terms by size and then add pairs of neighbour terms together repeatedly in a merging pattern. This way it is it can avoid a lot of cases where numbers of very different sizes are added together.

(f) In the information appearing when typing help(sum) it is said that the behaviour related to overflow can be platform dependent, at least with values of different signs.

Problem 2

The computations done for this problem are shown in code 3. Note that the number of calculations are not that large and the results may give a wrong impression on the speed of floating point computation speed versus integer computation speed. Sometimes the timing will heavily depend on whether we want to force the returned value to be an integer. The results are presented in table 1.

Table 1: Comparison of timing for float variables and integers.

Operation	Fl. time/s	Int. time/s
Division	0.64	11.11
Subset	0.14	0.12
Change entries	0.02	0.03
Addition	14.04	4.64
Scalar mult.	0.33	8.02

Code 3: Comparison of timing for float variables and integers.

```
# Initializing a numeric vector and an integer vector in order to do timing of
# operations on these
bf = rep(2, 1e8)
bi = as.integer(rep(2, 1e8))

# Doing vectorized divisions
divFtime = unname(system.time(cf <- bf/2)[3]) # 0.64
divItime = unname(system.time(cFromI <- bi/2)[3]) # 11.11
# Creating subsets of the vectors
subsFtime = unname(system.time(df <- bf[seq(1, length(bf), 50)])[3]) # 0.14
subsItime = unname(system.time(di <- as.integer(bf[seq(1, length(bi), 50)]))[3]) # 0.12
# Changing some elements in the vectors
changeFtime = unname(system.time(df[seq(1, length(df), 5)] <- 3)[3]) # 0.02
changeItime = unname(system.time(di[seq(1, length(di), 5)] <- as.integer(3))[3]) # 0.03

# Adding vectors together
addFtime = unname(system.time(ef <- bf + bf)[3]) # 14.04
addItime = unname(system.time(ei <- bi + bi)[3]) # 4.64

# Scalar multiplied with vector
multFtime = unname(system.time(ff <- 10*bf)[3]) # 0.33
multItime = unname(system.time(fi <- as.integer(10*bi))[3]) # 8.02
```