Problem Set 5

Thibault Doutre, ID : 26980469

STAT 243 : Introduction to Statistical Computing

I worked on my own.

1 Problem 1

1.1 Part a

If we store 1.000000000001 on a computer, we have 16 digits of accuracy. We can see it by setting the "digits" option in R.

```
options(digits=16)
1.000000000001

## [1] 1.000000000001

options(digits=17)
1.000000000001

## [1] 1.0000000000010001
```

1.2 Part b

```
options(digits=16)
x = c(1,rep(1e-16,10000))
sum(x)

## [1] 1.000000000001
```

The sum function gives the right answer, with the corresponding accuracy of 16 digits.

1.3 Part c

In python however, sum does not give the right answer. We can see that the equivalent of setting digits=16 option in R is setting '.15f' option in python.

```
import numpy as np
x=np.concatenate(([1],np.repeat(1e-16,10000)))
#sum
print(format(sum(x), '.15f'))
# precision 15f
print(format(1.000000000001, '.15f'))
# precision 16f
print(format(1.000000000001, '.16f'))

## 1.000000000000000
## 1.000000000001000
## 1.0000000000010001
```

1.4 Part d

In R:

```
# ((x1 + x2) + x3) + ...
sum1=0
for (i in x)
  sum1=sum1+i
sum1

## [1] 1

# ((xn + xn-1) + xn-2) + ...
sum2=0
for (i in length(x):1)
  sum2=sum2+x[i]
sum2

## [1] 1.000000000001
```

Summing in the natural order does not give the right answer but adding in the reverse order gives the good precision.
In Python, same results:

```
import numpy as np
x=np.concatenate(([1],np.repeat(1e-16,10000)))
#((x1 + x2) + x3) + ...
sum1=0
for i in range(0,len(x)):
```

```
  sum1=sum1+x[i]
print(format(sum1, '.15f'))
#((xn + xn-1) + xn-2) + ...
sum2=0
for i in range(0,len(x)):
  sum2=sum2+x[len(x)-1-i]
print(format(sum2, '.15f'))

## 1.000000000000000
## 1.000000000001000
```

1.5 Part e

We can see that the sum function does not sum from the right to the left with this example.

```
x = c(rep(1e-16,10000),1)
# ((x1 + x2) + x3) + ...
sum1=0
for (i in x)
  sum1=sum1+i
sum1

## [1] 1.000000000001

# ((xn + xn-1) + xn-2) + ...
sum2=0
for (i in length(x):1)
  sum2=sum2+x[i]
sum2

## [1] 1
```

1.6 Part f

The sum function sums the numbers which have the same order of magnitude before adding them together.
Actually, sum is a primitive function : it calls C code directly with .Primitive() and contains no R code.

```
sum

## function (..., na.rm = FALSE)  .Primitive("sum")
```

2 Problem 2

Basic computations such as summing or subsetting are similar for floats and integers, but there are some computations which are faster when using foating numbers. And example of this is crossproduct, and for this case working with floats in much faster. I would recommand to use integers for sorting and summary though.
We can also notice that using the sum function works a bit faster for floats but using the Reduce('+',) it is similar for integers and for floating numbers.

```
library(rbenchmark)
options(digits = 4)
n=10000
integers=sample(1:n,n)
integers2=sample(1:n,n)
floats=sample(1:n+0.0,n)
floats2=sample(1:n+0.0,n)

#sum
benchmark(
  int = sum(integers) ,
  float = sum(floats),
  replications = 100000,
  columns=c('test', 'elapsed', 'replications'))

##    test elapsed replications
## 2 float   0.953       100000
## 1   int   1.004       100000

#sumBis
benchmark(
  int = Reduce('+',integers) ,
  float = Reduce('+',floats),
  replications = 200,
  columns=c('test', 'elapsed', 'replications'))

##    test elapsed replications
## 2 float   0.652          200
## 1   int   0.644          200
```

```r
#subsetting
benchmark(
  int = integers[1:100] ,
  float = floats[1:100],
  replications = 10000,
  columns=c('test', 'elapsed', 'replications'))

##    test elapsed replications
## 2 float   0.025        10000
## 1   int   0.035        10000

#substracting
benchmark(
  int = integers-integers2 ,
  float = floats-floats2,
  replications = 10000,
  columns=c('test', 'elapsed', 'replications'))

##    test elapsed replications
## 2 float   0.155        10000
## 1   int   0.344        10000

#crossproduct
benchmark(
  int = crossprod(integers,integers2) ,
  float = crossprod(floats,floats2),
  replications = 10000,
  columns=c('test', 'elapsed', 'replications'))

##    test elapsed replications
## 2 float   0.104        10000
## 1   int   0.485        10000

#sort
benchmark(
  int = sort(integers) ,
  float = sort(floats),
  replications = 1000,
  columns=c('test', 'elapsed', 'replications'))

##    test elapsed replications
## 2 float   0.686         1000
## 1   int   0.629         1000
```

```
#summary
benchmark(
  int = summary(integers) ,
  float = summary(floats),
  replications = 1000,
  columns=c('test', 'elapsed', 'replications'))

##    test elapsed replications
## 2 float   0.600         1000
## 1   int   0.497         1000
```