STAT243 Problem Set 5

Name: Chih Hui Wang SID: 26955255

October 18, 2015

```
options(digits=22)
```

1. (a) The accuracy we should have is 16 places.
(b) Indeed, the accuracy of sum is 16 places.

```
#Create the vector
x <- c(1, rep(1e-16, 10000))

sum(x)

## [1] 1.000000000000999644811
```

(c) When using the sum in Python, it loses the accuracy after the decimal point.

```
import numpy as np
import decimal as dm

# Create Vector
vec = np.array([1e-16]*(10001))
vec[0] = 1

v0 = dm.Decimal(sum(vec))
print(v0)

## 1
```

(d) For the loop in a reversed way, it gives us the correct accuracy while for the original order, it lose the accuracy.

```
#1 is the first term
value <- 0

for(i in 1:length(x)){
  value <- value + x[i]
}

#Ouput
value

## [1] 1
```

```r
#1 is the last term
value <- 0

for(i in length(x):1){
  value <- value + x[i]
}

#Ouput
value

## [1] 1.000000000001000088901
```

The same thing happens in Python.

```python
import numpy as np
import decimal as dm

# Create Vector
vec = np.array([1e-16]*(10001))
vec[0] = 1

# 1 is the first term
value = 0
index = np.arange(10000)

for i in index:
value = value + vec[i]

v1 = dm.Decimal(value)
print(v1)

## 1
```

```python
import numpy as np
import decimal as dm

# Create Vector
vec = np.array([1e-16]*(10001))
vec[0] = 1

# 1 is the last term
value = 0
index = np.arange(10000)

for i in reversed(index):
value = value + vec[i]

v2 = dm.Decimal(value)
print(v2)

## 1.0000000000009998668559774159803055226802825927734375
```

(e) The result of **sum** indicate that it address the problem smartly when there are some extremely small or large number. From previous result, it shows that the **sum** definitely does not use the **for** loop easily from the beginning to end. From the help page, it does not show too many details on it. We can use the **show_c_source** function in package **pryr** to look the source code of sum(**show_c_source(.Primitive(sum(x)))**).

2. I did three operation, for-loop sum, inverse matrix, subset, to compare the speed between integer and double.

First, I created a all 1 vector, **x**, and use for loop to get the sum of them. The speed for integer is quite similar to double.

```
#Change the output digits back to 7
options(digits=7)

#for loop
x <- rep(1, 10000000)

#Change data type
intx <- as.integer(x)
#Examine type
typeof(intx)

## [1] "integer"

ins <- 0
```

```
#Examine type
typeof(x)

## [1] "double"

s <- 0

library(rbenchmark)
#Comparison
benchmark(
  integer=for(i in 1:length(intx)){
    ins <- ins + intx[i]
  },
  double=for(i in 1:length(x)){
    s <- s + x[i]
  },
  replications=5
)

##      test replications elapsed relative user.self sys.self user.child
## 2  double            5  31.231    1.000    30.776    0.344          0
## 1 integer            5  33.560    1.075    33.480    0.048          0
##   sys.child
## 2         0
## 1         0
```

Second, the inverse of matrix, I created a big matrix and use **solve** to get its inverse. It turns that the speed for integer is faster than double.

```
# Inverse of matrix
# Integer
```

```r
intm <- matrix(sample(1:2000, size=1000000, replace=TRUE),
    ncol=1000, nrow=1000)

#Examine type
typeof(intm)

## [1] "integer"
```

```r
#Change data type
m <- apply(intm, 1, as.numeric)

#Examine type
typeof(m)

## [1] "double"

#Comparison
benchmark(
  integer=solve(intm),
  double=solve(m),
  replications=5
)

##      test replications elapsed relative user.self sys.self user.child
## 2  double            5   1.719    1.064     3.204    0.212          0
## 1 integer            5   1.615    1.000     3.056    0.152          0
##   sys.child
## 2         0
## 1         0
```

Finally, the subset, I created a vector **x** from 1 to 100000 and use **sample** to make a random index vector to subset **x**. The result is that integer performs faster than doouble.

```r
# Subsetting
# Integer
intx <- 1:1000000

#Examine type
typeof(intx)

## [1] "integer"
```

```r
#Change data type
x <- as.numeric(x)

#Examine type
typeof(x)

## [1] "double"

#Comparison
benchmark(
  integer=intx[sample(length(intx))],
```

```
  double=x[sample(length(x))],
  replications=5
)
```

```
##      test replications elapsed relative user.self sys.self user.child
## 2  double            5   4.519   22.044     4.396    0.124          0
## 1 integer            5   0.205    1.000     0.204    0.000          0
##   sys.child
## 2         0
## 1         0
```