Problem Set 5

Natalia Sarabia Vasquez

October 29, 2021

1 Problem 1

1. In class and in the Unit 6 notes, I mentioned that integers as large as 2^{53} can be stored exactly in the double precision floating point representation. ① Demonstrate how the integers 1, 2, 3, ...,$2^{53}-2$, $2^{53}-1$ can be stored exactly in the $(-1)^S \times 1.d \times 2^{e-1023}$ format where d is represented as 52 bits. I'm not expecting anything particularly formal - just write out for a few numbers and show the pattern. Then ② show that 2^{53} and $2^{53}+2$ can be represented exactly but $2^{53}+1$ cannot, so the spacing of numbers of this magnitude is 2. ③ Finally show that for numbers starting with 2^{54} that the spacing between integers that can be represented exactly is 4. (Note you don't need to write out e in base 2; you can use base 10). ④ Then confirm that what you've shown is consistent with the result of executing $2^{53}-1$, 2^{53}, and $2^{53}+1$ in R.

$$\underbrace{(-1)^S \times 1.d \times 2^{e-1023}}$$

① We can write:

$$1 = 2^0 \qquad (-1)^0 \times 1.\overbrace{000\,0\cdots0}^{52\text{ places.}} \times 2^0$$

$$2 = 2^1 \qquad (-1)^0 \times 1.000\cdots0 \times 2^1$$

$$3 = 2^1+2^0 \qquad (-1)^0 \times 1.100\cdots0 \times 2^1$$

$$4 = 2^2 \qquad (-1)^0 \times 1.00\cdots0 \times 2^2$$

$$5 = 2^2+2^0 \qquad (-1)^0 \times 1.10\cdots0 \times 2^2$$

$$6 = 2^2+2^1 \qquad (-1)^0 \times 1.10\cdots0 \times 2^2$$

$$7 = 2^2+2^1+2^0 \qquad (-1)^0 \times 1.110\cdots0 \times 2^2$$

I just realized how the pattern work for all the integers from 1 to $2^{53}-1$:

⋮

$$129 = 2^7+2^0 = (-1)^0 \times 1.0\,0\,0000\,1\,0\cdots0\ 2^7$$

$$2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$$

⋮

$$259 = \underset{256}{2^8} + 2^1 + 2^0 = (-1)^0 \times 1.00\,0\,0\,0\,0\,1\,1\,0\cdots0\,2^8$$

$$2^8\ 2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$$

⋮

Before I proceed, note this pattern:

$32 = 2^5 \longrightarrow$ can be expressed as a power of 2

$32 + 16 + 8 + 4 + 2 = 62$

$62 = 2^6 - 2 = 2^5 + 2^4 + 2^3 + 2^2 + 2^1$

$32 + 16 + 8 + 4 + 2 + 1 = 63$

$63 = 2^6 - 1 = 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0$ $\leftarrow$ the preceding number is the sum of all of the preceeding powers, all activated.

$64 = 2^6 \longrightarrow$ can be expressed as a power of 2

Same case here:

$64 + 32 + 16 + 8 + 4 + 2 = 126$

$126 = 2^7 - 2 = 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1$

$64 + 32 + 16 + 8 + 4 + 2 + 1 = 127$

$127 = 2^7 - 1 = 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0$

$128 = 2^7$

So, we have:

$$\vdots$$

$$2^{53} - 2 = (-1)^0 \underbrace{1.111\cdots10}_{\text{I activate all the places but the last one.}} \times 2^{52}$$

$$2^{53} - 1 = (-1)^0 \underset{2^{52}}{1}.\underset{2^{51}}{1}11\cdots\underset{2^{1}}{1}\underset{2^{0}}{1} \times 2^{52}$$

As a conclusion, I just showed how the integers from 1 to $2^{53}-1$ can be stored exactly in the:

$$(-1)^{s} \times 1.d \times 2^{e-1023}$$

format, where d is represented as 52 bits.

② How can I represent 2^{53}, $2^{53}+1$ and $2^{53}+2$?

$$2^{53} = (-1)^{0} \times 1.00\cdots0 \times 2^{53}$$

$$2^{53}+2 = (-1)^{0} \times \underset{\substack{\downarrow \\ 2^{53}}}{1}.0\cdots\underset{\substack{\downarrow \\ 2^{1}}}{1} \times 2^{53}$$

What happens to $2^{53}+1$?

$$2^{53}+1 = (-1)^{0} \times 1.0\cdots0\underset{\substack{\downarrow \\ 2^{1}}}{1} \times 2^{53}$$

I will need to have 2^{0} as the last bit, but as we only have 52 positions available and the last one is 2^{1}, I cannot represent $2^{53}+1$ exactly.

Therefore, I was able to represent 2^{53} and $2^{53}+2$ but no $2^{53}+1$, exactly. That means that somehow $2^{53}+1$ will be rounded to one number that it is close to it and that has actual representation.

So, the space between numbers we can represent is: $2^{53}+2-2^{53}=2$.

3 Finally, for numbers starting with 2^{54} the spacing is 4.

$2^{54} = 1.0 \dots 0 \times 2^{54}$

$2^{54}+1 = 1.0 \dots 0$ (2^{54} 2^{53} 2^{2}) — we cannot represent this number exactly, I need a 2^0 but the minimum is 2^2

$2^{54}+2 = 1.0 \dots$ (2^{54} 2^{53} $\dots$ 2^{2}) — we cannot represent this number exactly, I need a 2^1 but the minimum is 2^2

$2^{54}+3 = 1.0 \dots$ — Again, I need to "activate" the positions 2^1 and 2^0 but I can't.

$2^{54}+4 = 1.0 \dots 1 \times 2^{54}$ ($\downarrow 2^{54}$ $\downarrow 2^{53}$ $\dots \downarrow 2^{2}$) — I can represent this number exactly!

Therefore, the spacing between integers that can be represented is:

$$2^{54} + 4 - 2^{54} = 4$$

$\textcircled{4}$ Confirm in R that what you've shown is consistent with the result of executing $2^{53}-1$, 2^{53} and $2^{53}+1$.

What I am about to see is that 2^{53} and $2^{53}-1$ can be represented exactly and therefore, if I perform the substraction the difference between them will be one.

As $2^{53}+1$ cannot be represented exactly it gets rounded to the nearest integer and therefore the substraction of $2^{53}+1-2^{53}$ is zero (Actually, both have the same bite-wise representation).

```
# I will use the professor's function to format the output

## Define a wrapper function for convenience:
```

```r
dg <- function(x, digits = 20) formatC(x, digits, format = 'f')
```

Confirm in R that what you have shown is consistent:

```r
a <- 2^53
bits(2^53)
```

```
## [1] "01000011 01000000 00000000 00000000 00000000 00000000 00000000 00000000"
```

```r
b <- (2^53)+1
bits((2^53)+1)
```

```
## [1] "01000011 01000000 00000000 00000000 00000000 00000000 00000000 00000000"
```

```r
c <- (2^53)-1
bits((2^53)-1)
```

```
## [1] "01000011 00111111 11111111 11111111 11111111 11111111 11111111 11111111"
```

```r
# Note that actually a and b have the same bit-wise representation
# That's why the difference is zero
dg(b-a)
```

```
## [1] "0.00000000000000000000"
```

```r
# c can be represented exactly, therefore the difference between the two numbers is  not zero
dg(a-c)
```

```
## [1] "1.00000000000000000000"
```

```r
# We can see the decimal representations for the numbers here:
dg(a)
```

```
## [1] "9007199254740992.00000000000000000000"
```

```r
dg(b)
```

```
## [1] "9007199254740992.00000000000000000000"
```

```r
dg(c)
```

```
## [1] "9007199254740991.00000000000000000000"
```

2 Problem 2

2.1 Problem 2a

How many digits of accuracy do we have? In general, we expect to have 16 digits of accuracy. In this particular example, if we store 1.000000000001, we expect to have 16 digits of accuracy, which means, we will have all the digits of the correct result (13 digits) plus three trailing zeroes, which result in 16 digits of accuracy.

2.2 Problem 2b

In R, does the use of sum() give the right answer up to the accuracy expected from part (a)? Yes, it gives the expected accuracy (16 digits). We have to stop just before the 6, and round. This will give us 16 digits of accuracy.

```r
x <- c(1,rep(1e-16,10000))
dg(sum(x))
```

```
## [1] "1.00000000000099964481"
```

2.3 Problem 2c

In Python, does the use of sum() give the right answer up to the accuracy expected from part (a)?

No, it doesn't. I got 15 digits of accuracy. You have to stop in the last nine in order that, after rounding, obtain the correct answer. This means that we have 15 digits of accuracy.

```
import numpy as np
from decimal import *

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

# Default is 28 places
Decimal(np.sum(vec))
```

```
## Decimal('1.0000000000009985345883478657924570143222808837890625')
```

2.4 Problem 2d

For loop to do the summation, R. Does this give the right answer? If either of these don't give the right answer, how many decimal places of accuracy does the answer have?

```
x <- c(1,rep(1e-16,10000))
aux = x[1]

for(i in seq_len(10000)){
  aux = aux + x[i+1]
}

dg(aux)
```

```
## [1] "1.00000000000000000000"
```

For loop to do the summation, R. 1 as the last value of the vector:

```
x <- c(rep(1e-16,10000),1)
aux = x[1]

for(i in seq_len(10000)){
  aux = aux + x[i+1]
}

dg(aux)
```

```
## [1] "1.00000000000100008890"
```

In R, doing the calculation taking 1 as the first number of the sum gives us a wrong result. In this case, we have 12 digits of accuracy. When we take 1 as the last number of the sum, we get the 16 digits of accuracy. This is a clear example how computer arithmetic does not obey the associative and distributive laws.

For loop to do the summation, Python:

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

aux = vec[0]
```

```
for i in range(1,10001):
  aux = aux + vec[i]

Decimal(aux)
```

```
## Decimal('1')
```

For loop to do the summation, Python. 1 as the last value of the vector:

```
vec = np.array([1e-16]*(10001))
vec[10000] = 1

aux = vec[0]

for i in range(0,10001):
  aux = aux + vec[i]

Decimal(aux)
```

```
## Decimal('1.0000000000010000889005823410116136074066162109375')
```

In Python, doing the calculation taking 1 as the first number of the sum gives us a wrong result. In this case, we have 12 digits of accuracy if we consider the non-visible zeroes. When we take 1 as the last number of the sum, we get the 16 digits of accuracy.

3 Problem 3

3.1 Problem 3a

In my assigned SCF Linux server arwen, I:

i. Copied the files to a subdirectory called tmp.

```
# Login in my SCF account
ssh natalia_sarabia10@arwen.berkeley.edu
# Create a directory called tmp in my SCF
mkdir tmp
# Locate at the recent created directory
cd tmp
# Once there, copy all the files to my tmp directory in SCF
scp /scratch/users/paciorek/wikistats/dated_2017_small/dated/*.gz* .
```

ii. Wrote efficient R code to, using foreach with the doFuture backend that, in parallel, read in the space delimited files and filter to only the rows that refer to pages where “Barack_Obama” appears in the page title.

I did some extra steps to test my code before running with all the files. First, I moved two files to my local machine, and test my code sequentially in one of them. Once everything was working, I parallelized over the two. Finally, I was ready to run my whole code in the Linux server:

```
############# Optional task ##############
# This was not mandatory, but I did this as a good practice.
# Back in my local machine...
# Copy some of the files (2) to make some tests using R in my local machine

# CAUTION: I will break the following two lines just to format.
# However, if I want to execute this, those breaklines should not be considered:
```

```
scp natalia_sarabia10@arwen.berkeley.edu:~/tmp/part-00116.gz*
  ~/Documents/STAT243/PS5/part-00116.gz*

scp natalia_sarabia10@arwen.berkeley.edu:~/tmp/part-00117.gz*
  ~/Documents/STAT243/PS5/part-00117.gz*

# Execute my code sequentially and the parallelization for two files (all this in R)
# Once I know everything is working as expected, run the parallelization on my SCF machine
########################################

# First, pass the R script (MyScript.R) to my SCF directory
scp MyScript.R natalia_sarabia10@arwen.berkeley.edu:~/tmp/

# Again, in my SCF account...
# Locate myself in temp
# I have to give the file permission to execute
chmod u+x MyScript.R

############# Optional task ##############
## First, check that it runs in R
# Initialize R
R
# Execute my modified script. Just for two files:
source('MyScript.R')
# Check in tmp, I must have my ouput Obama_matches.csv
# First, I kill R
q()
# Once out of there, check my files
ls
less Obama_matches.csv
########################################

###################### Working with all the files on my SCF machine
## Now that I checked that it works fine in R, I can execute the whole
# First, locate myself in tmp
cd tmp

# Run the script
R CMD BATCH MyScript.R

# Send the output to my local machine
# I have to be in my local machine to run this
scp natalia_sarabia10@arwen.berkeley.edu:~/tmp/Obama* ~/Documents/STAT243/PS5/.
```

Here is my script, MyScript.R

```
# Required libraries
library(future)
library(doFuture)
library(foreach)
library(dplyr)
library(readr)
library(stringr)
library(lubridate)
```

```r
library(ggplot2)
library(scales)

# Set-up
nFiles <- 191
filesPath <- "part-"
filesNames <- sprintf("%05d", seq(from = 0, to = nFiles, by = 1))

nCores <- 4
plan(multisession, workers = nCores)
registerDoFuture()

# Read the files, keep only lines we are interested on
# Using a foreach, I read the files. I tell the column type to be more efficient.
# Finally, I filter by the websites that have the "Barack_Obama" string
result <- foreach(i = seq_len(nFiles)) %dopar% {
  output <- read_delim(paste0(filesPath,filesNames[i],".gz"),
                       delim = " ", quote="\"",
                       col_names = c("date", "time", "language", "webpage",
                                     "n_hits", "page_size"),
                       col_types = cols(date = "i",
                                        time = "i",
                                        language = "c",
                                        webpage = "c",
                                        n_hits = "i",
                                        page_size = "d")) %>%
    filter(str_detect(webpage, "Barack_Obama"))
}

# Collect the results in a data frame
result <- data.frame(Reduce(rbind, result))

# Plot results
# I took care of the hour format
result %>%
  mutate(hour = round(time/10000),
         date = ISOdate(2008,11,04,hour,0,0,tz="GMT")) %>%
  group_by(date) %>%
  summarise(n_hits = sum(n_hits,na.rm = TRUE)/1000) %>%
  ggplot(aes(date,n_hits)) +
  geom_line(linetype = "dotted", size = 0.30) +
  geom_point() +
  theme_bw() +
  scale_x_datetime(date_breaks = '1 hours',labels = date_format("%H:%M",tz = "GMT")) +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1)) +
  labs(title = "Number of hits on websites containing 'Barack_Obama'. Nov 4, 2008") +
  theme(plot.title = element_text(size=10)) +
  xlab("Hours") + ylab("Number of hits (thousands)")

# Save the plot
ggsave("Obama.png")

# I will also have a .csv to show that the task was actually executed
```

```r
write_csv(result, file = "Obama.csv")
```

iii. My R script generates a .csv with the collected data. It also generates the plot. I transferred both outputs to my local machine to show that my script works properly.

```r
Obama <- read_csv(file.path('..','PS5/Obama.csv')) %>%
  mutate(hour = round(time/10000),
         date = ISOdate(2008,11,04,hour,0,0,tz="GMT")) %>%
  group_by(date) %>%
  summarise(n_hits = sum(n_hits,na.rm = TRUE)/1000)

head(Obama, n = 10)
```

```
## # A tibble: 10 x 2
##    date                n_hits
##    <dttm>               <dbl>
##  1 2008-11-04 00:00:00   39.3
##  2 2008-11-04 01:00:00   45.7
##  3 2008-11-04 02:00:00   38.8
##  4 2008-11-04 03:00:00   37.5
##  5 2008-11-04 04:00:00   35.5
##  6 2008-11-04 05:00:00   30.3
##  7 2008-11-04 06:00:00   25.8
##  8 2008-11-04 07:00:00   24.0
##  9 2008-11-04 08:00:00   26.2
## 10 2008-11-04 09:00:00   27.5
```

```r
include_graphics(file.path('..','PS5/Obama.png'))
```

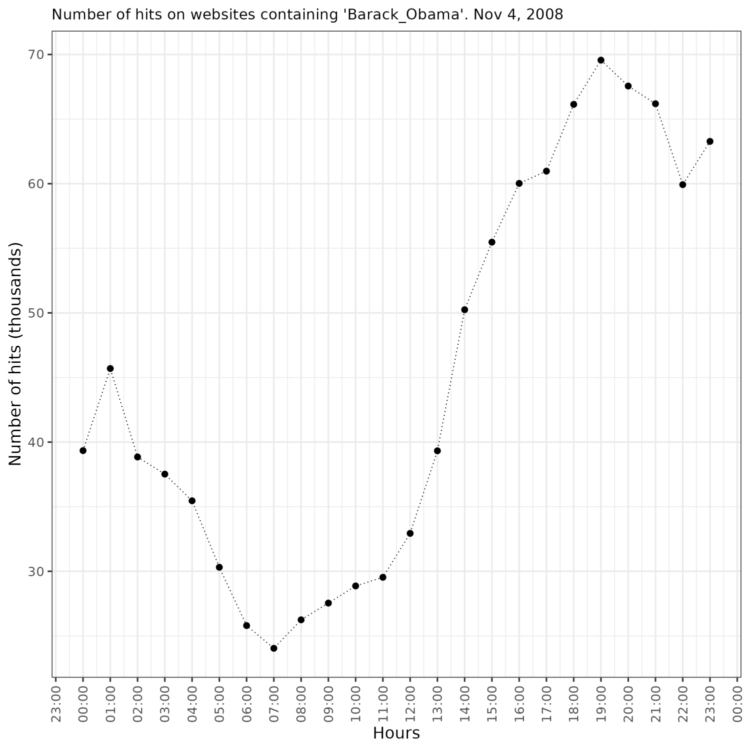
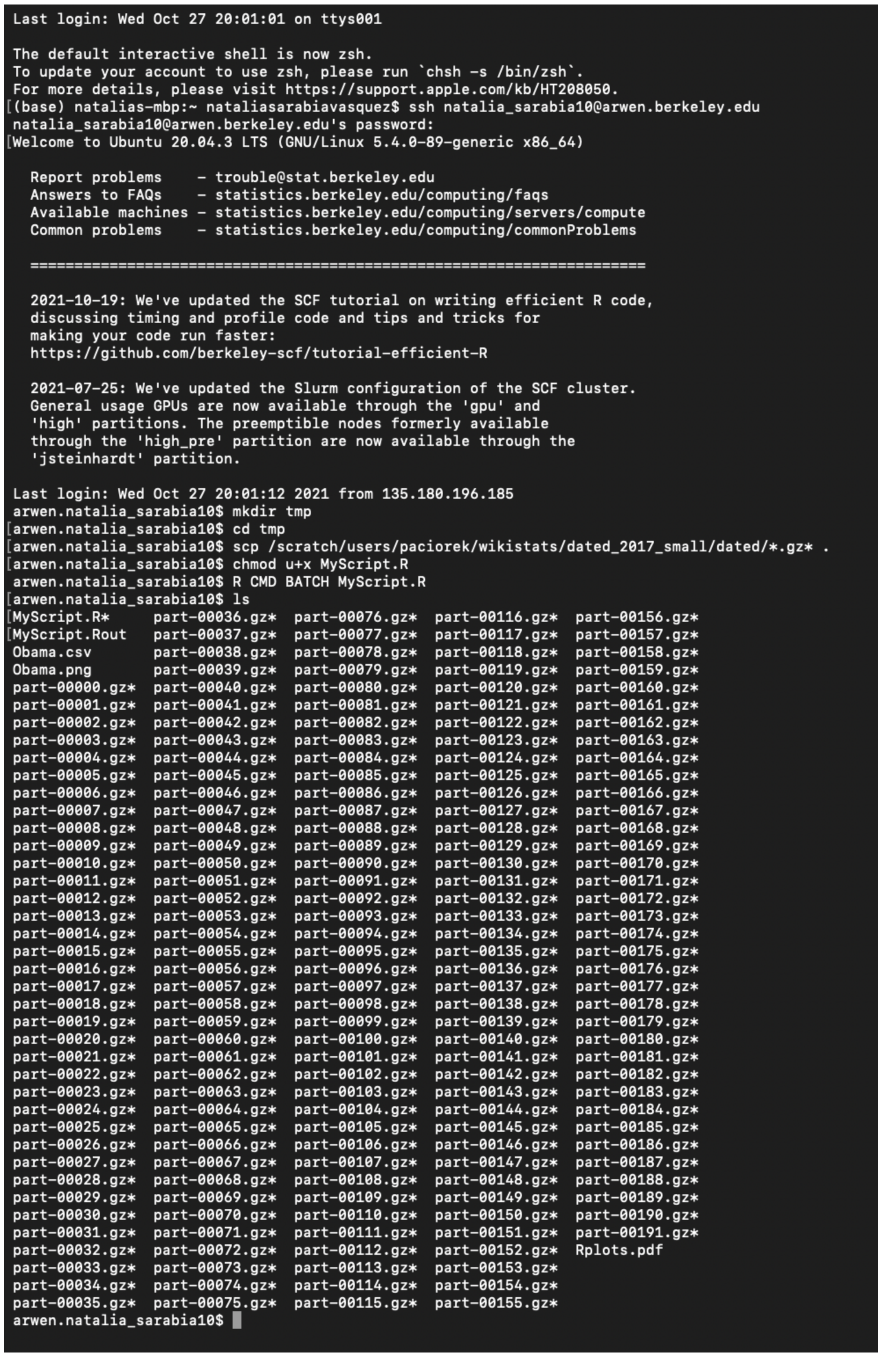


Figure 1: Hits to websites containing ’Barack Obama’

I show what I ran in my SCF account:

```
include_graphics(file.path('..','PS5/P3a.png'))
```



```
Last login: Wed Oct 27 20:01:01 on ttys001

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
[(base) natalias-mbp:~ nataliasarabiavasquez$ ssh natalia_sarabia10@arwen.berkeley.edu
natalia_sarabia10@arwen.berkeley.edu's password:
[Welcome to Ubuntu 20.04.3 LTS (GNU/Linux 5.4.0-89-generic x86_64)

  Report problems    - trouble@stat.berkeley.edu
  Answers to FAQs    - statistics.berkeley.edu/computing/faqs
  Available machines - statistics.berkeley.edu/computing/servers/compute
  Common problems    - statistics.berkeley.edu/computing/commonProblems

  ======================================================================

  2021-10-19: We've updated the SCF tutorial on writing efficient R code,
  discussing timing and profile code and tips and tricks for
  making your code run faster:
  https://github.com/berkeley-scf/tutorial-efficient-R

  2021-07-25: We've updated the Slurm configuration of the SCF cluster.
  General usage GPUs are now available through the 'gpu' and
  'high' partitions. The preemptible nodes formerly available
  through the 'high_pre' partition are now available through the
  'jsteinhardt' partition.

Last login: Wed Oct 27 20:01:12 2021 from 135.180.196.185
arwen.natalia_sarabia10$ mkdir tmp
[arwen.natalia_sarabia10$ cd tmp
[arwen.natalia_sarabia10$ scp /scratch/users/paciorek/wikistats/dated_2017_small/dated/*.gz* .
[arwen.natalia_sarabia10$ chmod u+x MyScript.R
arwen.natalia_sarabia10$ R CMD BATCH MyScript.R
[arwen.natalia_sarabia10$ ls
[MyScript.R*     part-00036.gz*  part-00076.gz*  part-00116.gz*  part-00156.gz*
[MyScript.Rout   part-00037.gz*  part-00077.gz*  part-00117.gz*  part-00157.gz*
Obama.csv        part-00038.gz*  part-00078.gz*  part-00118.gz*  part-00158.gz*
Obama.png        part-00039.gz*  part-00079.gz*  part-00119.gz*  part-00159.gz*
part-00000.gz*   part-00040.gz*  part-00080.gz*  part-00120.gz*  part-00160.gz*
part-00001.gz*   part-00041.gz*  part-00081.gz*  part-00121.gz*  part-00161.gz*
part-00002.gz*   part-00042.gz*  part-00082.gz*  part-00122.gz*  part-00162.gz*
part-00003.gz*   part-00043.gz*  part-00083.gz*  part-00123.gz*  part-00163.gz*
part-00004.gz*   part-00044.gz*  part-00084.gz*  part-00124.gz*  part-00164.gz*
part-00005.gz*   part-00045.gz*  part-00085.gz*  part-00125.gz*  part-00165.gz*
part-00006.gz*   part-00046.gz*  part-00086.gz*  part-00126.gz*  part-00166.gz*
part-00007.gz*   part-00047.gz*  part-00087.gz*  part-00127.gz*  part-00167.gz*
part-00008.gz*   part-00048.gz*  part-00088.gz*  part-00128.gz*  part-00168.gz*
part-00009.gz*   part-00049.gz*  part-00089.gz*  part-00129.gz*  part-00169.gz*
part-00010.gz*   part-00050.gz*  part-00090.gz*  part-00130.gz*  part-00170.gz*
part-00011.gz*   part-00051.gz*  part-00091.gz*  part-00131.gz*  part-00171.gz*
part-00012.gz*   part-00052.gz*  part-00092.gz*  part-00132.gz*  part-00172.gz*
part-00013.gz*   part-00053.gz*  part-00093.gz*  part-00133.gz*  part-00173.gz*
part-00014.gz*   part-00054.gz*  part-00094.gz*  part-00134.gz*  part-00174.gz*
part-00015.gz*   part-00055.gz*  part-00095.gz*  part-00135.gz*  part-00175.gz*
part-00016.gz*   part-00056.gz*  part-00096.gz*  part-00136.gz*  part-00176.gz*
part-00017.gz*   part-00057.gz*  part-00097.gz*  part-00137.gz*  part-00177.gz*
part-00018.gz*   part-00058.gz*  part-00098.gz*  part-00138.gz*  part-00178.gz*
part-00019.gz*   part-00059.gz*  part-00099.gz*  part-00139.gz*  part-00179.gz*
part-00020.gz*   part-00060.gz*  part-00100.gz*  part-00140.gz*  part-00180.gz*
part-00021.gz*   part-00061.gz*  part-00101.gz*  part-00141.gz*  part-00181.gz*
part-00022.gz*   part-00062.gz*  part-00102.gz*  part-00142.gz*  part-00182.gz*
part-00023.gz*   part-00063.gz*  part-00103.gz*  part-00143.gz*  part-00183.gz*
part-00024.gz*   part-00064.gz*  part-00104.gz*  part-00144.gz*  part-00184.gz*
part-00025.gz*   part-00065.gz*  part-00105.gz*  part-00145.gz*  part-00185.gz*
part-00026.gz*   part-00066.gz*  part-00106.gz*  part-00146.gz*  part-00186.gz*
part-00027.gz*   part-00067.gz*  part-00107.gz*  part-00147.gz*  part-00187.gz*
part-00028.gz*   part-00068.gz*  part-00108.gz*  part-00148.gz*  part-00188.gz*
part-00029.gz*   part-00069.gz*  part-00109.gz*  part-00149.gz*  part-00189.gz*
part-00030.gz*   part-00070.gz*  part-00110.gz*  part-00150.gz*  part-00190.gz*
part-00031.gz*   part-00071.gz*  part-00111.gz*  part-00151.gz*  part-00191.gz*
part-00032.gz*   part-00072.gz*  part-00112.gz*  part-00152.gz*  Rplots.pdf
part-00033.gz*   part-00073.gz*  part-00113.gz*  part-00153.gz*
part-00034.gz*   part-00074.gz*  part-00114.gz*  part-00154.gz*
part-00035.gz*   part-00075.gz*  part-00115.gz*  part-00155.gz*
arwen.natalia_sarabia10$
```

Figure 2: Part a

3.2 Problem 3b

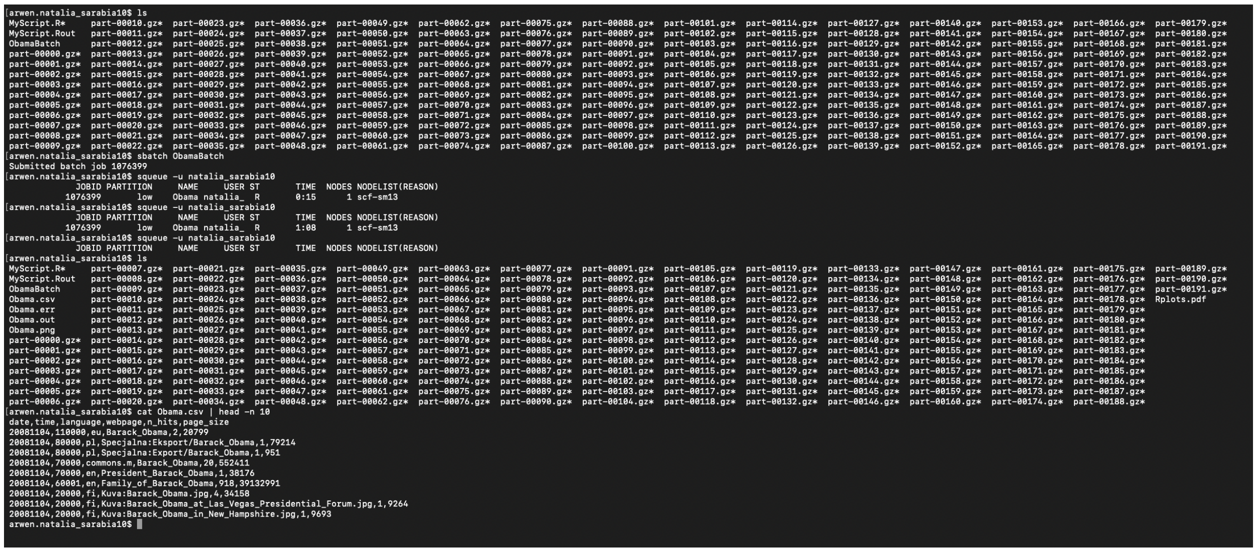
My ObamaBatch file looks like:

```
#!/bin/bash

####################
# SBATCH OPTIONS
####################
#SBATCH --job-name=Obama                    # job name fore queue, default may be u$
#SBATCH --partition=low                     # high/low/gpu, default if empty is low
#SBATCH --error=Obama.err                   # error file, default if empty is slurm$
#SBATCH --output=Obama.out                  # standard out file, no default
#SBATCH --time=00:15:00                     # optional, max runtime of job hrs:min:sec
#SBATCH --nodes=1                           # only use 1 node, MPI option
#SBATCH --ntasks=1                          # how many tasks to start
#SBATCH --cpus-per-task=4                   # number of cores to use, multi-core/mu$

####################
# What to run
####################

scp /scratch/users/paciorek/wikistats/dated_2017_small/dated/*.gz* temp/
R CMD BATCH MyScript.R MyScript.Rout
```

```
################ Part b ################
# First, in my local machine, modify the ObamaBatch file:
# Set a limit time and the number of cores, job name, standard out and error out
# files. Also the run time of the job, and the number of cores to use.
# The file must have the line including the instruction to copy the files

# Save, change name and send it to your SCF account
scp ObamaBatch natalia_sarabia10@arwen.berkeley.edu:~/tmp/

# Execute the file in your SCF account
sbatch ObamaBatch

# Monitor your job
squeue -u natalia_sarabia10

# Check that the file contains the desired output
cat Obama.csv | head -n 10

# Check that the code was actually executed -> Obama's files created:
ls
```

I add a screenshot of my command line while running the code above. First, I removed all the files created in 3a and then I executed my R script, using sbatch again. I monitored the job. When it finished, I check that my directory contained the desired output files, and it does. They have the desired output:

```
include_graphics(file.path('..','PS5/P3b.png'))
```



```
[arwen.natalia_sarabia10$ ls
MyScript.R*     part-00010.gz*  part-00023.gz*  part-00036.gz*  part-00049.gz*  part-00062.gz*  part-00075.gz*  part-00088.gz*  part-00101.gz*  part-00114.gz*  part-00127.gz*  part-00140.gz*  part-00153.gz*  part-00166.gz*  part-00179.gz*
MyScript.Rout   part-00011.gz*  part-00024.gz*  part-00037.gz*  part-00050.gz*  part-00063.gz*  part-00076.gz*  part-00089.gz*  part-00102.gz*  part-00115.gz*  part-00128.gz*  part-00141.gz*  part-00154.gz*  part-00167.gz*  part-00180.gz*
ObamaBatch      part-00012.gz*  part-00025.gz*  part-00038.gz*  part-00051.gz*  part-00064.gz*  part-00077.gz*  part-00090.gz*  part-00103.gz*  part-00116.gz*  part-00129.gz*  part-00142.gz*  part-00155.gz*  part-00168.gz*  part-00181.gz*
part-00000.gz*  part-00013.gz*  part-00026.gz*  part-00039.gz*  part-00052.gz*  part-00065.gz*  part-00078.gz*  part-00091.gz*  part-00104.gz*  part-00117.gz*  part-00130.gz*  part-00143.gz*  part-00156.gz*  part-00169.gz*  part-00182.gz*
part-00001.gz*  part-00014.gz*  part-00027.gz*  part-00040.gz*  part-00053.gz*  part-00066.gz*  part-00079.gz*  part-00092.gz*  part-00105.gz*  part-00118.gz*  part-00131.gz*  part-00144.gz*  part-00157.gz*  part-00170.gz*  part-00183.gz*
part-00002.gz*  part-00015.gz*  part-00028.gz*  part-00041.gz*  part-00054.gz*  part-00067.gz*  part-00080.gz*  part-00093.gz*  part-00106.gz*  part-00119.gz*  part-00132.gz*  part-00145.gz*  part-00158.gz*  part-00171.gz*  part-00184.gz*
part-00003.gz*  part-00016.gz*  part-00029.gz*  part-00042.gz*  part-00055.gz*  part-00068.gz*  part-00081.gz*  part-00094.gz*  part-00107.gz*  part-00120.gz*  part-00133.gz*  part-00146.gz*  part-00159.gz*  part-00172.gz*  part-00185.gz*
part-00004.gz*  part-00017.gz*  part-00030.gz*  part-00043.gz*  part-00056.gz*  part-00069.gz*  part-00082.gz*  part-00095.gz*  part-00108.gz*  part-00121.gz*  part-00134.gz*  part-00147.gz*  part-00160.gz*  part-00173.gz*  part-00186.gz*
part-00005.gz*  part-00018.gz*  part-00031.gz*  part-00044.gz*  part-00057.gz*  part-00070.gz*  part-00083.gz*  part-00096.gz*  part-00109.gz*  part-00122.gz*  part-00135.gz*  part-00148.gz*  part-00161.gz*  part-00174.gz*  part-00187.gz*
part-00006.gz*  part-00019.gz*  part-00032.gz*  part-00045.gz*  part-00058.gz*  part-00071.gz*  part-00084.gz*  part-00097.gz*  part-00110.gz*  part-00123.gz*  part-00136.gz*  part-00149.gz*  part-00162.gz*  part-00175.gz*  part-00188.gz*
part-00007.gz*  part-00020.gz*  part-00033.gz*  part-00046.gz*  part-00059.gz*  part-00072.gz*  part-00085.gz*  part-00098.gz*  part-00111.gz*  part-00124.gz*  part-00137.gz*  part-00150.gz*  part-00163.gz*  part-00176.gz*  part-00189.gz*
part-00008.gz*  part-00021.gz*  part-00034.gz*  part-00047.gz*  part-00060.gz*  part-00073.gz*  part-00086.gz*  part-00099.gz*  part-00112.gz*  part-00125.gz*  part-00138.gz*  part-00151.gz*  part-00164.gz*  part-00177.gz*  part-00190.gz*
part-00009.gz*  part-00022.gz*  part-00035.gz*  part-00048.gz*  part-00061.gz*  part-00074.gz*  part-00087.gz*  part-00100.gz*  part-00113.gz*  part-00126.gz*  part-00139.gz*  part-00152.gz*  part-00165.gz*  part-00178.gz*  part-00191.gz*
[arwen.natalia_sarabia10$ sbatch ObamaBatch
Submitted batch job 1076399
[arwen.natalia_sarabia10$ squeue -u natalia_sarabia10
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
           1076399       low    Obama natalia_  R       0:15      1 scf-sm13
[arwen.natalia_sarabia10$ squeue -u natalia_sarabia10
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
           1076399       low    Obama natalia_  R       1:08      1 scf-sm13
[arwen.natalia_sarabia10$ squeue -u natalia_sarabia10
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
[arwen.natalia_sarabia10$ ls
MyScript.R*     part-00007.gz*  part-00021.gz*  part-00035.gz*  part-00049.gz*  part-00063.gz*  part-00077.gz*  part-00091.gz*  part-00105.gz*  part-00119.gz*  part-00133.gz*  part-00147.gz*  part-00161.gz*  part-00175.gz*  part-00189.gz*
MyScript.Rout   part-00008.gz*  part-00022.gz*  part-00036.gz*  part-00050.gz*  part-00064.gz*  part-00078.gz*  part-00092.gz*  part-00106.gz*  part-00120.gz*  part-00134.gz*  part-00148.gz*  part-00162.gz*  part-00176.gz*  part-00190.gz*
ObamaBatch      part-00009.gz*  part-00023.gz*  part-00037.gz*  part-00051.gz*  part-00065.gz*  part-00079.gz*  part-00093.gz*  part-00107.gz*  part-00121.gz*  part-00135.gz*  part-00149.gz*  part-00163.gz*  part-00177.gz*  part-00191.gz*
Obama.csv       part-00010.gz*  part-00024.gz*  part-00038.gz*  part-00052.gz*  part-00066.gz*  part-00080.gz*  part-00094.gz*  part-00108.gz*  part-00122.gz*  part-00136.gz*  part-00150.gz*  part-00164.gz*  part-00178.gz*  Rplots.pdf
Obama.err       part-00011.gz*  part-00025.gz*  part-00039.gz*  part-00053.gz*  part-00067.gz*  part-00081.gz*  part-00095.gz*  part-00109.gz*  part-00123.gz*  part-00137.gz*  part-00151.gz*  part-00165.gz*  part-00179.gz*
Obama.out       part-00012.gz*  part-00026.gz*  part-00040.gz*  part-00054.gz*  part-00068.gz*  part-00082.gz*  part-00096.gz*  part-00110.gz*  part-00124.gz*  part-00138.gz*  part-00152.gz*  part-00166.gz*  part-00180.gz*
Obama.png       part-00013.gz*  part-00027.gz*  part-00041.gz*  part-00055.gz*  part-00069.gz*  part-00083.gz*  part-00097.gz*  part-00111.gz*  part-00125.gz*  part-00139.gz*  part-00153.gz*  part-00167.gz*  part-00181.gz*
part-00000.gz*  part-00014.gz*  part-00028.gz*  part-00042.gz*  part-00056.gz*  part-00070.gz*  part-00084.gz*  part-00098.gz*  part-00112.gz*  part-00126.gz*  part-00140.gz*  part-00154.gz*  part-00168.gz*  part-00182.gz*
part-00001.gz*  part-00015.gz*  part-00029.gz*  part-00043.gz*  part-00057.gz*  part-00071.gz*  part-00085.gz*  part-00099.gz*  part-00113.gz*  part-00127.gz*  part-00141.gz*  part-00155.gz*  part-00169.gz*  part-00183.gz*
part-00002.gz*  part-00016.gz*  part-00030.gz*  part-00044.gz*  part-00058.gz*  part-00072.gz*  part-00086.gz*  part-00100.gz*  part-00114.gz*  part-00128.gz*  part-00142.gz*  part-00156.gz*  part-00170.gz*  part-00184.gz*
part-00003.gz*  part-00017.gz*  part-00031.gz*  part-00045.gz*  part-00059.gz*  part-00073.gz*  part-00087.gz*  part-00101.gz*  part-00115.gz*  part-00129.gz*  part-00143.gz*  part-00157.gz*  part-00171.gz*  part-00185.gz*
part-00004.gz*  part-00018.gz*  part-00032.gz*  part-00046.gz*  part-00060.gz*  part-00074.gz*  part-00088.gz*  part-00102.gz*  part-00116.gz*  part-00130.gz*  part-00144.gz*  part-00158.gz*  part-00172.gz*  part-00186.gz*
part-00005.gz*  part-00019.gz*  part-00033.gz*  part-00047.gz*  part-00061.gz*  part-00075.gz*  part-00089.gz*  part-00103.gz*  part-00117.gz*  part-00131.gz*  part-00145.gz*  part-00159.gz*  part-00173.gz*  part-00187.gz*
part-00006.gz*  part-00020.gz*  part-00034.gz*  part-00048.gz*  part-00062.gz*  part-00076.gz*  part-00090.gz*  part-00104.gz*  part-00118.gz*  part-00132.gz*  part-00146.gz*  part-00160.gz*  part-00174.gz*  part-00188.gz*
[arwen.natalia_sarabia10$ cat Obama.csv | head -n 10
date,time,language,webpage,n_hits,page_size
20081104,110000,eu,Barack_Obama,2,20799
20081104,80000,pl,Specjalna:Eksport/Barack_Obama,1,79214
20081104,80000,pl,Specjalna:Export/Barack_Obama,1,951
20081104,70000,commons.m,Barack_Obama,20,552411
20081104,70000,en,President_Barack_Obama,1,38176
20081104,60001,en,Family_of_Barack_Obama,918,39132991
20081104,20000,fi,Kuva:Barack_Obama.jpg,4,34158
20081104,20000,fi,Kuva:Barack_Obama_at_Las_Vegas_Presidential_Forum.jpg,1,9264
20081104,20000,fi,Kuva:Barack_Obama_in_New_Hampshire.jpg,1,9693
arwen.natalia_sarabia10$ 
```

Figure 3: Part b

4 Problem 4

a) What are the goals of their simulation study and what are the metrics that they consider in assessing their method?

The researchers wanted to investigate finite samples properties of the estimators that were derived in earlier sections of their paper (Section 2 and 3) using Monte Carlo simulation. They studied time invariant and time-dependent coefficients. They also performed a comparison with last value carried forward method. The metrics employed to evaluate the accuracies of the asymptotic approximations were the empirical bias ('Bias'), the bias divided by the true β_1 ('RB') and the sample standard deviation ('SD'). They also calculated a model-based standard error and the corresponding 0.95 confidence interval based on the normal approximation, that is, the average of the standard error estimates ('SE') and the coverage probability of the 95% confidence interval for β_1.

b) What choices did the authors have to make in designing their simulation study? What are the key aspects of the data generating mechanism that might affect their assessment of their method?

In the time invariant coefficient framework: They have to decide (set fixed) different aspects. For instance, the number of generated datasets, the number of subjects in each of them, the values for β_1 and β_0 (although they mentioned that they got similar results for different values of β_S). They also chose the Epanechnikov kernel, although they mentioned that similar results are obtained by using other kernels. In the case of the time-dependent coefficient, the simulation set-up is identical. Similarly, they chose to use some bandwiths. They also considered a wide range of functional forms for their comparison with last value carried forward method. In their work, they also talk about some distributions and specific parameters of them. Their distributions, given knowledge of the subjects, "generate" data that is similar to the one I can obtain in real life if I measure the problem.

The key aspects that can might affect their assessment of the method are the parameters of the distributions used to generate the data, the distributions they used (Poisson(5), Uniform). Also, the number of replications, the models employed and the sample sizes.

c) Consider their Tables 1 and 3 reporting the simulation results. For a method to be a good method, what would one want to see numerically in these columns?

I would like to see that as the sample size increases the bias decreases. That the empirical and model based SE tend to be similar. In general, that the performance of the estimator increases when the sample size increase. I am also looking for SE and SD small and that the coverage probability will be close to 95%, if this does not happen in many repetitions, the interval is not capturing the true parameter. From the tables of the document, I see that the time invariant coefficient and the time-dependent coefficient, most of these properties are satisfied, although there are some marginal differences. However, when comparing to the last value carried forward method, everything changes. Here the bias is considerable and the performance does not improves as the sample size increases. Also, as the variance decreases as n increases, the coverage probability decreases also and is not near the 95% level.

5 Extra credit

a) By experimentation in R, find the base 10 representation of the smallest positive number that can be represented in R.

In class, we saw that the smallest exponent that we can have is -1022, therefore the smallest positive normalized number is 1 * 2ˆ-1022 ≈ 2.2 * 10ˆ-308 (.Machine$double.xmin). Let's start with this approach:

```
.Machine$double.xmin
```

```
## [1] 2.225074e-308
```

```
# The binary representation makes sense. Our number has an exponent of -1022. If we
# see closer, it appears that this is the smallest representation of the exponent.
# But what if all the bits of the exponent are zeroes?
bits(2^-1022)
```

```
## [1] "00000000 00010000 00000000 00000000 00000000 00000000 00000000 00000000"
```

```
# What happens when all the bits of the exponents are zeroes? Can this number be
# represented in a computer?
bits(2^-1023)
```

```
## [1] "00000000 00001000 00000000 00000000 00000000 00000000 00000000 00000000"
```

```
# If the answer is yes, then we would have to work with their exponents in order to
# solve our problem:
2^-1024
```

```
## [1] 5.562685e-309
```

```
# WOW! It turns out that it has an actual representation... how far I can go?
# Well, if I had a fixed exponent of -1022, maybe I can go until -1074 (51's zeroes
# followed by a one, which corresponds to 2^-52. Therefore (2^-52)*(2^-1022) =
# (2^-1074)). Let's see:
2^-1074
```

```
## [1] 4.940656e-324
```

```
bits(2^-1074)
```

```
## [1] "00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000001"
```

```
# Approximately, in base 10:  4.940656e-324 -> 4.9 * 10^-324
2^-1074
```

```
## [1] 4.940656e-324
```

```
# What happens if I try to go even smaller, I guess it will underflow, but let's see.
# Until here, was pure logic, let's just try one more case, just to see if something
# new happens:
2^-1075
```

```
## [1] 0
```

```r
bits(2^-1075)
```

```
## [1] "00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000"
```

```r
# I overflowed. So, everything checks, the smallest number I can represent is
# 2^-1074 or 4.9 * 10^-324 in base 10. I have to be careful with the language. This #
# number is not normalized (meaning it does not take into account the number one at
# the beginning, the one that we do not need to represent in bits), that's why it
# is different from the smallest normalized number.
```

b) Explain how it can be that we can store a number smaller than 1 * 2^-1022.

Well, I guess the intuition was given in a. In the floating point representation there are normalized and not normalized representations for the numbers. This means, that for not normalized numbers, all the bits of my exponent will be zero. Also, the "initial" one we were taking for the normalized case will be zero in this case.

Following the points given in the instructions (that actually were followed in a):

```r
# Smallest number we saw in class:
bits(2^-1022)
```

```
## [1] "00000000 00010000 00000000 00000000 00000000 00000000 00000000 00000000"
```

```r
# We also have a bit-wise representation for 2^-1023:
# Decimal
2^-1023
```

```
## [1] 1.112537e-308
```

```r
bits(2^-1023)
```

```
## [1] "00000000 00001000 00000000 00000000 00000000 00000000 00000000 00000000"
```

```r
# If everything would have ended here, the bit-wise representation would have been
# just zeroes (as when we try to represent bits (2^-1075)), and we would have expect
# an underflow and the well-known zero. However, that is not happening and actually
# we are able to represent even smaller numbers.

# It is interesting that all the digits in the exponent are zeroes. So this idea is
# the biggest clue, in my opinion. We can have smaller numbers if we set all the bits
# in the exponent to be zeroes, what means an exponent of 2^-1022 multiplied by
# 2^-something. This results in adding the exponents and that means smaller numbers
# representation. The 'something' part is given by 2^-a, and a goes from left to
# right and increases, in magnitude, by one every place.

2^-1024
```

```
## [1] 5.562685e-309
```

```r
bits(2^-1024)
```

```
## [1] "00000000 00000100 00000000 00000000 00000000 00000000 00000000 00000000"
```

```r
# I can continue this logic, another example:
2^-1030
```

```
## [1] 8.691695e-311
```

```r
# And it has 'activated' the 8th place as expected, the rest are zeroes:
bits(2^-1030)
```

```
## [1] "00000000 00000000 00010000 00000000 00000000 00000000 00000000 00000000"
```

```r
# Therefore, following all the ideas from a and b so far, the smallest number
# must be all zeroes, except the last one. In other words, 51's zeroes followed by a
# one.
# That would be 2^-1022 multiplied by 2^-52, that gives us 2^-1074!

# Decimal representation: 4.940656e-324 -> 4.9 * 10^-324
2^-1074
```

```
## [1] 4.940656e-324
```

```r
# Bits representation
bits(2^-1074)
```

```
## [1] "00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000001"
```

```r
# As we saw before, everything ends with 2^-1075. Here, R cannot represent
# anything smaller (64 bit-wise representation constrain). Then, I finally underflow:
bits(2^-1075)
```

```
## [1] "00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000"
```

```r
2^-1075
```

```
## [1] 0
```