Homework 4

Advanced Statistical Computing (STAT 6984)

Sumin Shen

Problem 1: Profiling (15 pts)

Below are two, very compact, versions of functions for calculating powers of a matrix.

a. First, briefly explain the thought process behind these two new methods.

The function powers3 calculate the power of the vector x elements to the degree specified by the vector y elements.

The function powers4 first do a pre-allocation since number of rows and cols of matrix is known, same step in the function powers2. Then, use the function apply, for each row of the pre-allocated matrix, apply the function cumprod, which returns a vector whose elements are the cumulative products. The function cumprod does the operation on each row, but returned it in a column vector form, so the output from the function powers4 needed to be transformed before return.

b. Then provide a summary of the computation time for all four versions (two from lecture and the two above). Use the same `x` from lecture.

```
##         user.self sys.self elapsed
## powers1     0.472    0.088   0.561
## powers2     0.076    0.016   0.092
## powers3     1.144    0.016   1.159
## powers4     2.320    0.056   2.375
```

c. Profile the code to explain why the two new versions disappoint relative to the original two. Cite particular subroutines which cause the slowdowns with reference to the profile summaries. Are these subroutines they creating memory or computational bottlenecks, or both?

This is the profile for the function powers1 for reference. The function cbind takes most of the time and memory.

```
## $by.self
##         self.time self.pct total.time total.pct mem.total
## "cbind"      0.34      100       0.34       100     884.6
##
## $by.total
##                       total.time total.pct mem.total self.time self.pct
## "cbind"                     0.34       100     884.6      0.34      100
## "block_exec"                0.34       100     884.6      0.00        0
## "call_block"                0.34       100     884.6      0.00        0
## "eval"                      0.34       100     884.6      0.00        0
## "evaluate"                  0.34       100     884.6      0.00        0
## "evaluate_call"             0.34       100     884.6      0.00        0
## "handle"                    0.34       100     884.6      0.00        0
## "in_dir"                    0.34       100     884.6      0.00        0
```

```
## "knitr::knit"               0.34       100     884.6      0.00         0
## "powers1"                   0.34       100     884.6      0.00         0
## "process_file"              0.34       100     884.6      0.00         0
## "process_group"             0.34       100     884.6      0.00         0
## "process_group.block"       0.34       100     884.6      0.00         0
## "rmarkdown::render"         0.34       100     884.6      0.00         0
## "timing_fn"                 0.34       100     884.6      0.00         0
## "withCallingHandlers"       0.34       100     884.6      0.00         0
## "withVisible"               0.34       100     884.6      0.00         0
##
## $sample.interval
## [1] 0.02
##
## $sampling.time
## [1] 0.34
```

This is the profile for the function powers2 for reference. The function matrix takes some time, but less memory and time compared to the function powers1.

```
## $by.self
##           self.time self.pct total.time total.pct mem.total
## "cbind"        0.18       90       0.18        90     778.2
## "powers1"      0.02       10       0.20       100     778.2
##
## $by.total
##                       total.time total.pct mem.total self.time self.pct
## "powers1"                   0.20       100     778.2      0.02       10
## "block_exec"                0.20       100     778.2      0.00        0
## "call_block"                0.20       100     778.2      0.00        0
## "eval"                      0.20       100     778.2      0.00        0
## "evaluate"                  0.20       100     778.2      0.00        0
## "evaluate_call"             0.20       100     778.2      0.00        0
## "handle"                    0.20       100     778.2      0.00        0
## "in_dir"                    0.20       100     778.2      0.00        0
## "knitr::knit"               0.20       100     778.2      0.00        0
## "process_file"              0.20       100     778.2      0.00        0
## "process_group"             0.20       100     778.2      0.00        0
## "process_group.block"       0.20       100     778.2      0.00        0
## "rmarkdown::render"         0.20       100     778.2      0.00        0
## "timing_fn"                 0.20       100     778.2      0.00        0
## "withCallingHandlers"       0.20       100     778.2      0.00        0
## "withVisible"               0.20       100     778.2      0.00        0
## "cbind"                     0.18        90     778.2      0.18       90
##
## $sample.interval
## [1] 0.02
##
## $sampling.time
## [1] 0.2
```

```
## $by.self
##           self.time self.pct total.time total.pct mem.total
## "powers2"      0.06      100       0.06       100      68.7
##
## $by.total
##                       total.time total.pct mem.total self.time self.pct
## "powers2"                   0.06       100      68.7      0.06      100
## "block_exec"                0.06       100      68.7      0.00        0
## "call_block"                0.06       100      68.7      0.00        0
## "eval"                      0.06       100      68.7      0.00        0
## "evaluate"                  0.06       100      68.7      0.00        0
## "evaluate_call"             0.06       100      68.7      0.00        0
## "handle"                    0.06       100      68.7      0.00        0
## "in_dir"                    0.06       100      68.7      0.00        0
## "knitr::knit"               0.06       100      68.7      0.00        0
## "process_file"              0.06       100      68.7      0.00        0
## "process_group"             0.06       100      68.7      0.00        0
## "process_group.block"       0.06       100      68.7      0.00        0
## "rmarkdown::render"         0.06       100      68.7      0.00        0
## "timing_fn"                 0.06       100      68.7      0.00        0
## "withCallingHandlers"       0.06       100      68.7      0.00        0
## "withVisible"               0.06       100      68.7      0.00        0
##
## $sample.interval
## [1] 0.02
##
## $sampling.time
## [1] 0.06
```

This is the profile for the function powers3. The particular subroutines which cause the slowdowns with r are: outer. The function outer takes similar memory compared to the function powers1 but a lot more time.

```
## $by.self
##         self.time self.pct total.time total.pct mem.total
## "outer"      1.06      100       1.06       100     244.1
##
## $by.total
##                       total.time total.pct mem.total self.time self.pct
## "outer"                     1.06       100     244.1      1.06      100
## "block_exec"                1.06       100     244.1      0.00        0
## "call_block"                1.06       100     244.1      0.00        0
## "eval"                      1.06       100     244.1      0.00        0
## "evaluate"                  1.06       100     244.1      0.00        0
## "evaluate_call"             1.06       100     244.1      0.00        0
## "handle"                    1.06       100     244.1      0.00        0
## "in_dir"                    1.06       100     244.1      0.00        0
## "knitr::knit"               1.06       100     244.1      0.00        0
## "powers3"                   1.06       100     244.1      0.00        0
```

```
## "process_file"              1.06       100     244.1      0.00         0
## "process_group"             1.06       100     244.1      0.00         0
## "process_group.block"       1.06       100     244.1      0.00         0
## "rmarkdown::render"         1.06       100     244.1      0.00         0
## "timing_fn"                 1.06       100     244.1      0.00         0
## "withCallingHandlers"       1.06       100     244.1      0.00         0
## "withVisible"               1.06       100     244.1      0.00         0
##
## $sample.interval
## [1] 0.02
##
## $sampling.time
## [1] 1.06
```

This is the profile for the function powers4. The particular subroutine function apply takes 90% of the time. Compared to the other functions powers, it takes both the largest memory and time.

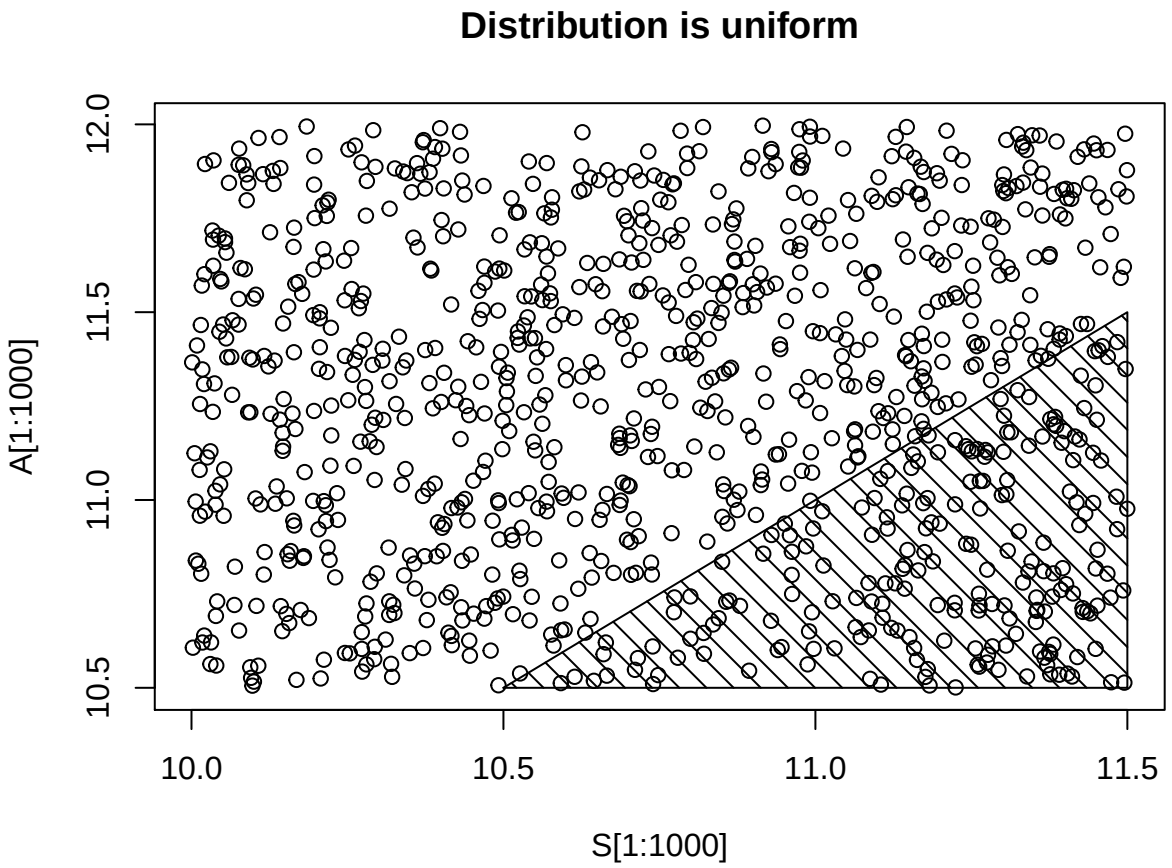
```
## $by.self
##                 self.time self.pct total.time total.pct mem.total
## "apply"              1.28    71.91       1.62     91.01    1198.8
## "t.default"          0.14     7.87       0.14      7.87     122.1
## "FUN"                0.12     6.74       0.12      6.74      71.4
## "unlist"             0.12     6.74       0.12      6.74     125.9
## "aperm.default"      0.06     3.37       0.06      3.37     122.1
## "array"              0.02     1.12       0.02      1.12     122.1
## "lengths"            0.02     1.12       0.02      1.12      19.6
## "matrix"             0.02     1.12       0.02      1.12       0.0
##
## $by.total
##                       total.time total.pct mem.total self.time self.pct
## "block_exec"                1.78    100.00    1320.8      0.00     0.00
## "call_block"                1.78    100.00    1320.8      0.00     0.00
## "eval"                      1.78    100.00    1320.8      0.00     0.00
## "evaluate"                  1.78    100.00    1320.8      0.00     0.00
## "evaluate_call"             1.78    100.00    1320.8      0.00     0.00
## "handle"                    1.78    100.00    1320.8      0.00     0.00
## "in_dir"                    1.78    100.00    1320.8      0.00     0.00
## "knitr::knit"               1.78    100.00    1320.8      0.00     0.00
## "powers4"                   1.78    100.00    1320.8      0.00     0.00
## "process_file"              1.78    100.00    1320.8      0.00     0.00
## "process_group"             1.78    100.00    1320.8      0.00     0.00
## "process_group.block"       1.78    100.00    1320.8      0.00     0.00
## "rmarkdown::render"         1.78    100.00    1320.8      0.00     0.00
## "timing_fn"                 1.78    100.00    1320.8      0.00     0.00
## "withCallingHandlers"       1.78    100.00    1320.8      0.00     0.00
## "withVisible"               1.78    100.00    1320.8      0.00     0.00
## "t"                         1.76     98.88    1320.8      0.00     0.00
## "apply"                     1.62     91.01    1198.8      1.28    71.91
## "t.default"                 0.14      7.87     122.1      0.14     7.87
```

```
## "FUN"                    0.12      6.74      71.4      0.12      6.74
## "unlist"                 0.12      6.74     125.9      0.12      6.74
## "aperm.default"          0.06      3.37     122.1      0.06      3.37
## "aperm"                  0.06      3.37     122.1      0.00      0.00
## "array"                  0.02      1.12     122.1      0.02      1.12
## "lengths"                0.02      1.12      19.6      0.02      1.12
## "matrix"                 0.02      1.12       0.0      0.02      1.12
##
## $sample.interval
## [1] 0.02
##
## $sampling.time
## [1] 1.78
```

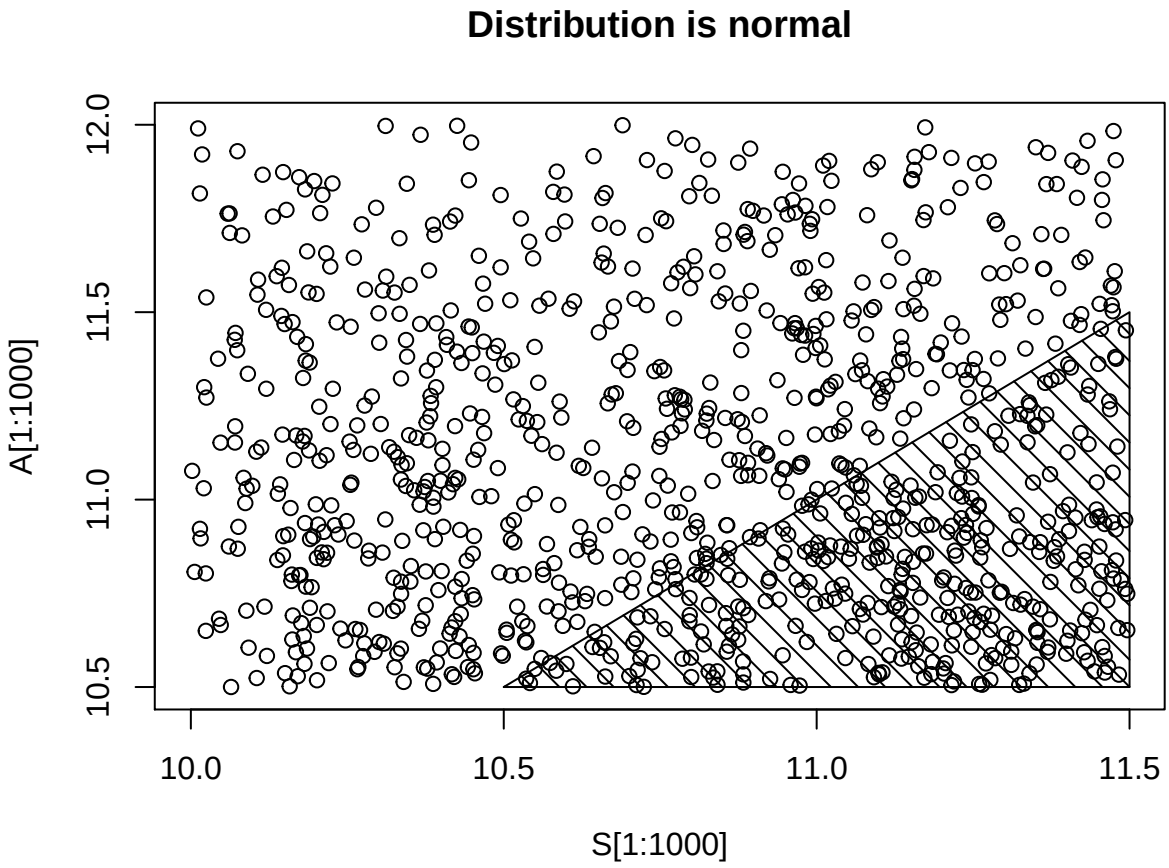
Problem 2: Annie & Sam in Lecture 8 (10 pts)

How would adjust the code for the Annie & Sam example(s) to accommodate other distributions? E.g., if $S \sim \mathcal{N}(10.5, 1)$ and $A \sim \mathcal{N}(11, 1.5)$?

```
# uniform case
out = simData(distType = "uniform")
```



```
# normal case
out = simData(distType = "normal"
              , par1_annie = 10.5, par2_annie = 1
              , par1_sam = 11, par2_sam = 1.5
       )
```



To adjust for other possible distributions, I wrote a function called simData in the file hw4_source.R. The function takes the argument, distType. Currently there are three choices: uniform, normal, and exponential. The argument distType provided by the user will be checked by the function match.arg. If the distribution is not in the supported pool distributions, error is generated. If there is no error, based on the value of the distType, random data will be generated.

For the case uniform distribution, no other arguments are needed except the argument distType. For the case normal distribution, additional parameters including mean and variance are needed. The data aregenerated from the truncated normal distribution.

Sam arrives at time between 10 and 11:30pm, and Annie arrives at time between 10:30 and 12am. For the normal distribution, the probability that Annie arrives before Sam is: 0.3008559, 0.3591441 with the 95% confidence interval as 0.3008559, 0.3591441. The expected difference is 0.28796 and absolute difference is 0.5292765, the respective CI's are 0.2513246, 0.3245954 and 0.5051149, 0.5534381.

Problem 3: Bootstrap with `boot` (15 pts)

Re-write the least-squares regression bootstrap from lecture using the `boot` library.

- Briefly compare and contrast to the results we obtained in lecture.

the mean matrix of the coef from the for-loop and the package boot are:

```
 apply(beta.hat.boot, MARGIN = 2, mean)
```

```
## [1]  0.806543305  1.993566634  3.018632080 -0.007472051
```

```
 apply(beta_boot_lib, MARGIN = 2, mean)
```

```
## [1]  0.819946679  1.994958138  3.017723695 -0.007564148
```

The covariance matrix of the coef from the for-loop and the package boot are:

```
cov_boot
```

```
##               [,1]          [,2]          [,3]          [,4]
## [1,]  0.0399538589 -6.143949e-04 -3.676180e-05  3.835386e-04
## [2,] -0.0006143949  6.535368e-04  1.170122e-05 -4.525274e-05
## [3,] -0.0000367618  1.170122e-05  2.207062e-04 -3.294129e-05
## [4,]  0.0003835386 -4.525274e-05 -3.294129e-05  1.266814e-04
```

```
cov_boot_lib
```

```
##              [,1]         [,2]         [,3]         [,4]
## [1,] 4.864966e-02 7.031845e-05 5.620552e-04 4.213408e-04
## [2,] 7.031845e-05 3.396044e-04 6.261383e-06 2.093986e-06
## [3,] 5.620552e-04 6.261383e-06 2.060611e-04 7.327214e-06
## [4,] 4.213408e-04 2.093986e-06 7.327214e-06 1.588989e-04
```

Figure shows the marginals of the sampling distributions from the for-loop methods.

Figure shows the marginals of the sampling distributions from the r package boot methods.

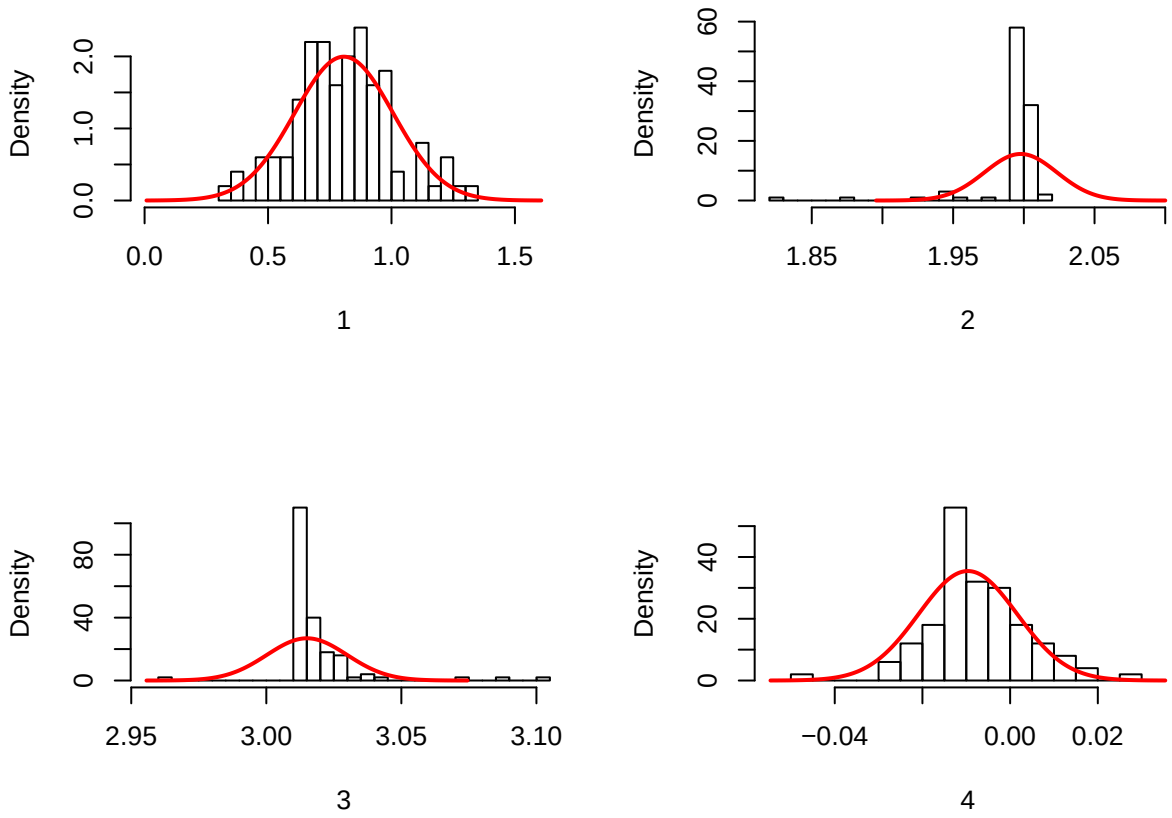
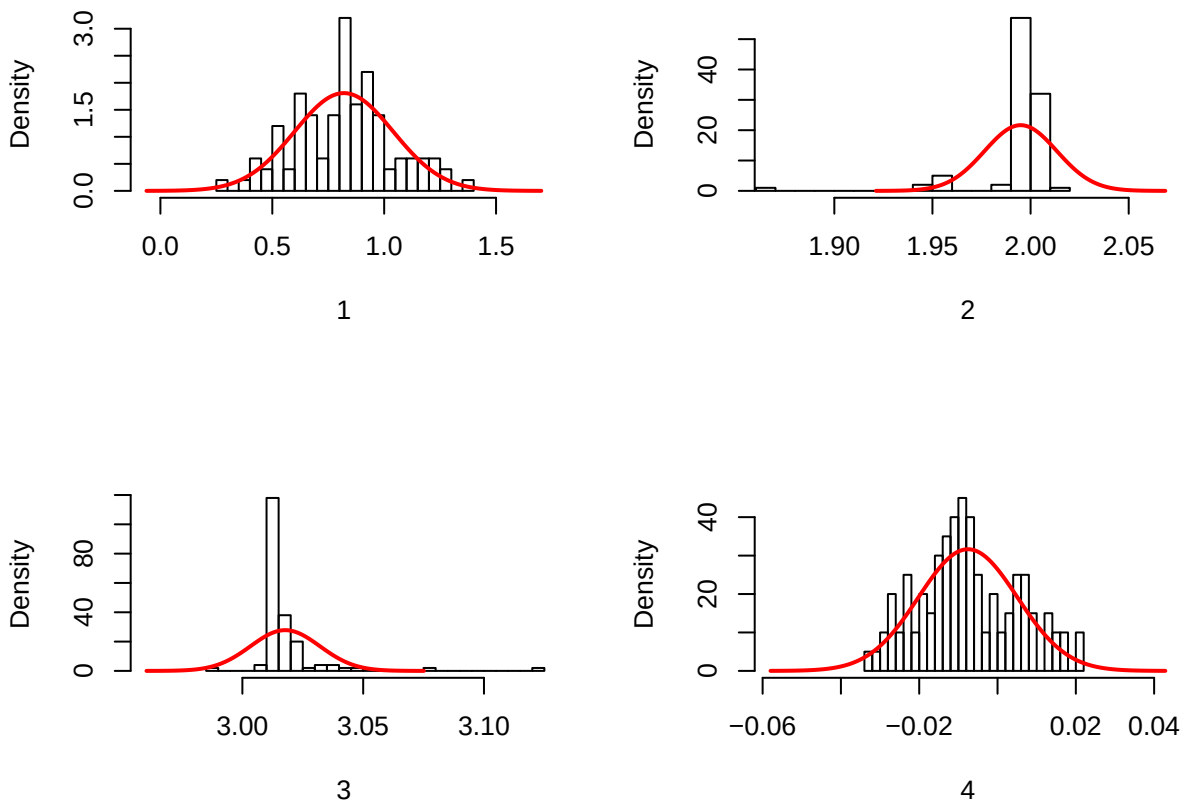


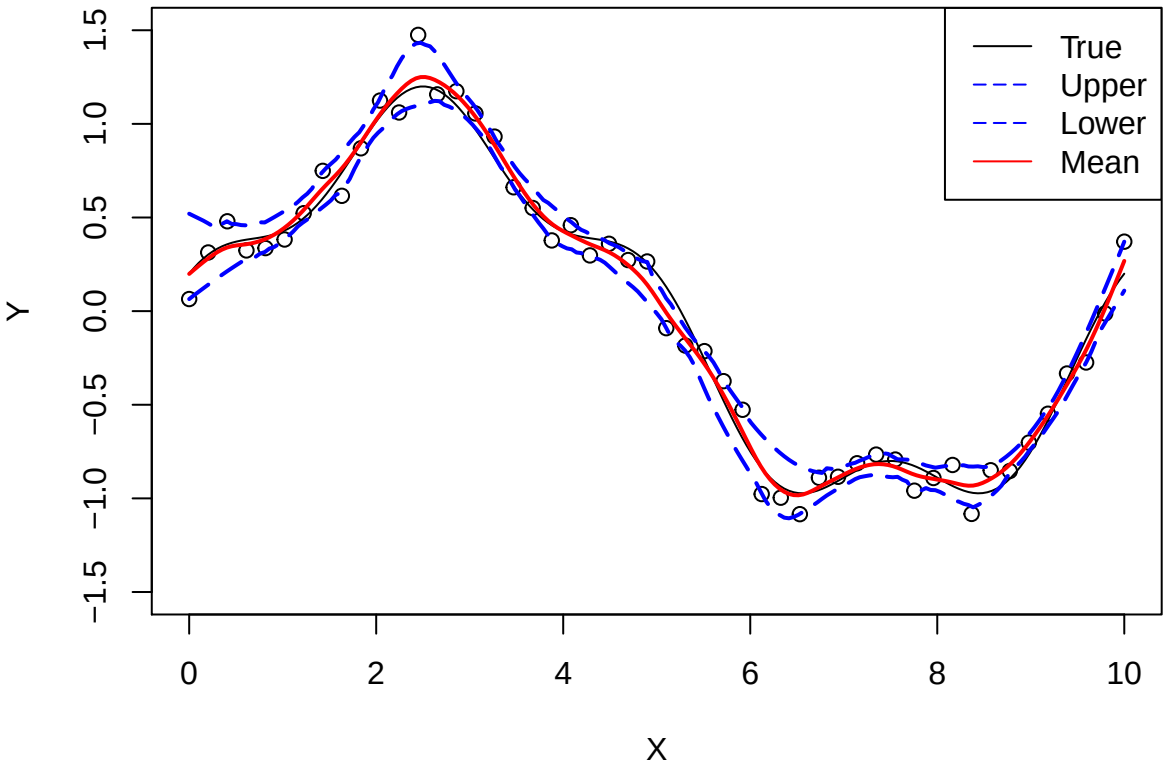
Figure 1: Bootstrap from the for-loop



Problem 4: Bootstrapped splines (15 pts)

Design a bootstrap to assess the predictive uncertainty in our {smooth.spline()} fits from slides 49–53 from stats.pdf.
Rather than specifying `df = 11`, use the CV option to fit the degrees of freedom. You may code the routine by hand, or within the `boot` library. Provide a visualization of the bootstrapped average predictive mean and central 90% quantiles.

The central 90% quantle is chosen such that the 5% quantile and 95% quantile are used as the lower and upper bounds, respectively.



Problem 5: Spam MC shell script (15 pts)

To run the script

```
./spam_mc.sh 1
```

Problem 6: Spam MC "bakeoff" in `parallel` (30 pts)

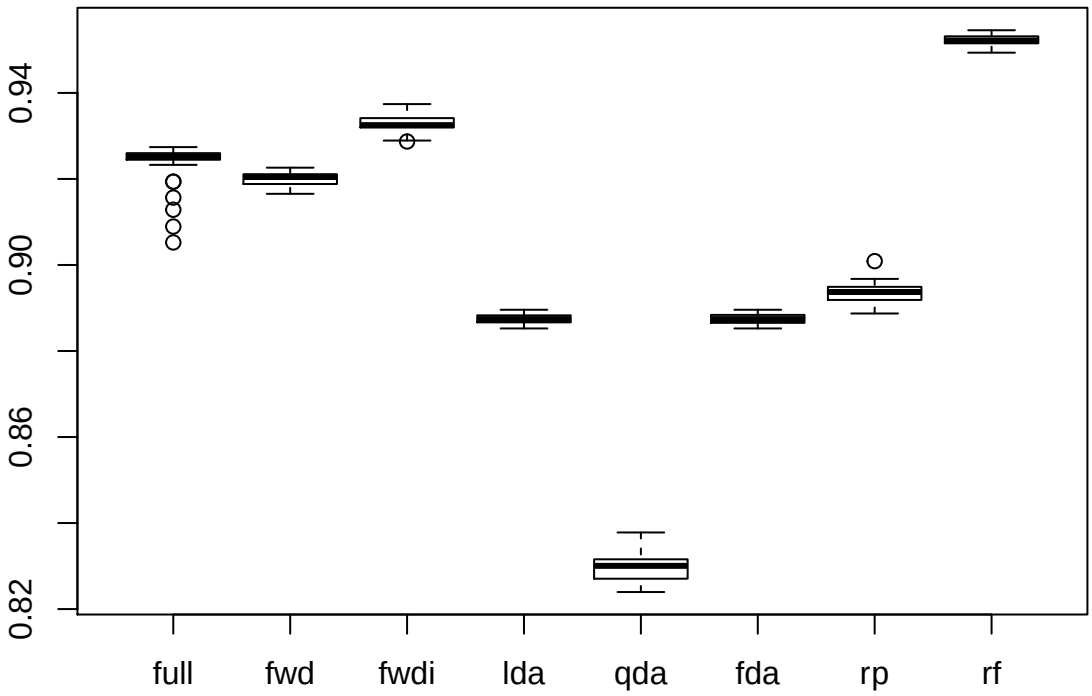
Re-write the spam MC "bakeoff" from lecture with sockets (i.e., via the `parallel` package) rather than via "batch mode".

In addition to summarizing the results of your experiment in your PDF writeup (these should be the same as in Problem 1, up to MC error), submitted via Canvas, you must also provide (via Bitbucket)

- An R script called `spam_snow.R` that runs the entire "bakeoff" using four parallel instances (sockets), by default.
- A shell script called `spam_snow.sh` that takes an integer command-line argument specifying the number of parallel instances (sockets) to create. Alternatively, you can make `spam_snow.R` directly executable (with the same command-line argument). Please indicate which in your solution and/or in a `README.md` file on Bitbucket. Whatever you choose, make sure to have a warning if the argument provided implies more instances than cores, as in Problem 5.

I ran the code and generate the results in parallel in Problem 6. 8 cores were used to run the scripts. So in total 40 repsetitions for the cross validation.

```
##  spam_1.RData spam_2.RData spam_3.RData spam_4.RData spam_5.RData spam_6.RData spam_7.RData
```




```
##         dfmean           tt
## rf   0.9522115 1.508996e-38
## fwdi 0.9327320 1.171411e-14
## full 0.9237177 7.666605e-06
## fwd  0.9200391 1.126057e-40
## rp   0.8935014 2.283682e-17
## fda  0.8875082 3.302156e-01
## lda  0.8875027 2.819784e-55
## qda  0.8295691 4.609335e-76
## null 0.6059552           NA
```

Problem ⋆: Spam summary

This isn't a real problem. It is just here as a place-holder to say that for Problems 5/6 you must demonstrate, in your PDF on Canvas, that you have been able to collect a substantial number of MC repetitions in order to re-visualize the results from lecture (which are only based on 5 repetitions). I expect at least thirty repetitions, which many would regard as a minimum number in order to "trust" the resulting accuracy distributions.

- Even with many instances in parallel, this will may take tens of hours so don't leave this until the last minute.
- If you need more computing power, please reach out to Steve to get access to our Linux servers.
- You don't need to do this fully for both 5 & 6, just one of them. Windows users may find 6 easier than 5 because to work in Bash will require the virtual machine, which is much slower. Mac users should be fine either way.
- Watched MCs don't work better than un-watched ones. (Like boiling water.) Work on one of the other, less computationally intensive, problems while these are running.

```
## [1] "/home/shen/Documents/VT/Classes/Fall_2017/StatisticalComputing/hw4"
```

```
## Session info -------------------------------------------------------------
```

```
##  setting  value
##  version  R version 3.4.2 (2017-09-28)
##  system   x86_64, linux-gnu
##  ui       X11
##  language en_US
##  collate  en_US.UTF-8
##  tz       America/New_York
##  date     2017-10-27
```

```
## Packages -----------------------------------------------------------------
```

```
##  package   * version date       source
##  backports   1.1.0   2017-05-22 CRAN (R 3.4.1)
##  base      * 3.4.2   2017-09-29 local
##  boot        1.3-20  2017-07-30 CRAN (R 3.4.2)
##  compiler    3.4.2   2017-09-29 local
##  datasets  * 3.4.2   2017-09-29 local
##  devtools    1.13.3  2017-08-02 CRAN (R 3.4.1)
##  digest      0.6.12  2017-01-27 CRAN (R 3.4.1)
##  evaluate    0.10.1  2017-06-24 CRAN (R 3.4.1)
##  graphics  * 3.4.2   2017-09-29 local
##  grDevices * 3.4.2   2017-09-29 local
##  highr       0.6     2016-05-09 CRAN (R 3.4.1)
##  htmltools   0.3.6   2017-04-28 CRAN (R 3.4.1)
##  knitr       1.16    2017-05-18 CRAN (R 3.4.1)
##  magrittr    1.5     2014-11-22 CRAN (R 3.4.1)
##  memoise     1.1.0   2017-04-21 CRAN (R 3.4.1)
##  methods   * 3.4.2   2017-09-29 local
##  Rcpp        0.12.12 2017-07-15 CRAN (R 3.4.1)
##  rmarkdown   1.6     2017-06-15 CRAN (R 3.4.1)
##  rprojroot   1.2     2017-01-16 CRAN (R 3.4.1)
##  splines   * 3.4.2   2017-09-29 local
##  stats     * 3.4.2   2017-09-29 local
##  stringi     1.1.5   2017-04-07 CRAN (R 3.4.1)
##  stringr     1.2.0   2017-02-18 CRAN (R 3.4.1)
##  tools       3.4.2   2017-09-29 local
##  truncnorm   1.0-7   2014-01-21 CRAN (R 3.4.2)
##  utils     * 3.4.2   2017-09-29 local
##  withr       2.0.0   2017-07-28 CRAN (R 3.4.1)
##  yaml        2.1.14  2016-11-12 CRAN (R 3.4.1)
```