STA 242 Assignment 5- New York City Taxi Data

Xueting Shao
git@bitbucket.org:sxshao/sta242.git

1. Introduction

This dataset was obtained through a Freedom of Information Law (FOIL) request from the New York City Taxi & Limousine Commission (NYCT&L). It covers four years (2010-2013) of taxi operations in New York City and includes 697,622,444 trips. The data is stored in CSV format, organized by year and month

There are 24 files of two groups: trip_data and trip_fare. In each file, each row represents a single taxi trip.

Trip_data: medallion, hack license,vender, rate_code, store_and_fwd_flag, pickup datetime, dropoff datetime, passenger count,trip time in secs, trip distance, pickup_longitude and pickup_latitude, dropoff longitude and dropoff latitude.

Trip_fare: medallion,hack license,vender id,pickup datetime,payment type, fare amount, surcharge, mta tax,tip amount,tolls amount,total amount.

Although the data is big and separate, but it is fun to find out the distribution of *total amount – tolls amount*, and build linear regression models to predict *total amount – tolls amount* using trip time/ trip time and surcharge.

2. Distribution of *Total Amount – Tolls Amount*

2.1. Boundary Values

Considering that the data is too big, it is worth getting some rough profiles of target variable *total amount – tolls amount*. Thus, boundary values are checked through all twelve trip_fare data and are shown in Table 1.

Table 1 Boundary Values

File	**1**	**2**	**3**	**4**
Max	650	540	529.8	615.69
Min	0	0	0	0
Number	14776616	13990177	15749229	15100469
File	**5**	**6**	**7**	**8**
Max	614.72	565.5	610.08	685908.1
Min	0	0	0	-1430
Number	15285050	14385457	13823841	12597110
File	**9**	**10**	**11**	**12**
Max	500	500	510	650
Min	0	0	0	0
Number	14107694	15004557	14388452	13971119

As we can see, most files have a max value around 500-650. File 8 has extremes values that reach to 685908.1 and -1430. Given the fact those should be outliers, I set the median of boundary values as corresponding overall boundary values for later discussion. Overall minimum would be 0 and maximum would be 587.79.

2.2. Deciles

Even a single data is hard to calculate the deciles, so I abstract frequency tables from each files and merge them. To be convenient, I arbitrarily set the same breaks for all the frequency tables.

In this case, I chose 0 as the minimum value and 587.79 as max value, which are obtained through the pre-checking boundary values. And the total number of breaks is 500 so the width of each interval would be (587.79-0)/500 = 1.17558. Aggregating all the frequency tables, the overall distribution is shown in Figure 1 and deciles are shown in Table 2.

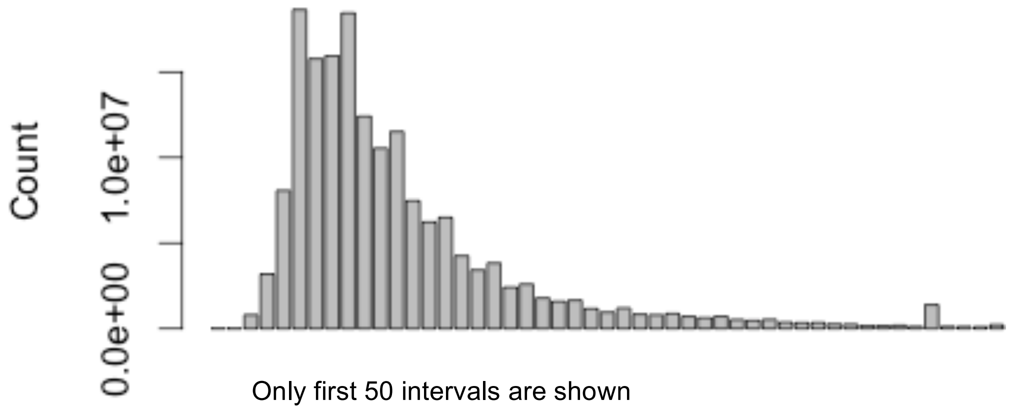


Figure 1 Histogram of All Files (Width = 1.17558)

Table 2 Deciles

0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
-1430	7.05	8.23	9.40	10.58	11.76	14.11	15.28	18.81	27.03	6.86×e5

From both histogram and deciles, the distribution of *total amount – tolls amount* has very long right tail and is seriously skewed right. It is not unexpected. Although in most cases, taxi drivers drive inside the city and get a fair pay, some drivers need to drive long distance or might meet a very generous client who gives a lot tips. The most interesting fact is the negative value. One possible reason is that the taxi driver broke the traffic law and got fined.

2.3. Different Approaches

Given the big size of data and limitation of computation and memory, it is worthwhile trying different approaches to obtain the deciles.

In the former discussion, there are three steps: (1) obtain frequency tables with the same scales from twelve files;(2) sum over the twelve tables into a general frequency table; (3) compute deciles from general frequency table. In this process, step (1):obtain frequency tables is the most complex and time-consuming. So I test three approaches.

- Approach 1: Parallel Process with only R Functions
- Approach 2: Parallel Process with C Routines
- Approach 3:Lapply Process with C Routines

And computing time is shown in Table 3.

Table 3 Computing Time

Approach 1(12 macines)		
User	**System**	**Elapsed**
10401.089	12.646	1470.479
Approach 2(12 machines)		
User	**System**	**Elapsed**
111.596	19.759	55.157
Approach 3		
User	**System**	**Elapsed**
223.000	34.888	399.065

Because the compute server is jam-packed, it is hard to compare computing time if I run three approaches one by one. I run three approaches simultaneously several time and get the similar computing time and the same results.(I know this will make the server suffer more but I have no choice) The approach using only R is extremely slower than approaches using C routines. But it is unexpected to see that elapsed time of approach 2 is only 6 times faster than approach 3 instead of 12. This may be caused by the limited memory and how the master allocates the task to 12 machines.

3. Regression

Then I want fit a regression model to predict *total amount – tolls amount* and trip time/surcharge is considered as the predictor variable. However, these three variables are not in one file. They are separated in trip_fare and trip_data, so it's uncertain all the rows in trip_fare and trip_data are matched. If they are not matched, then it is time-consuming to match them together to get a clean data frame. Trying to make programming simpler, I run my Check.R script to identify each row have the same medallion, hack license, vender ID, and pick up date time. Fortunately, all the rows are matched.

3.1. Simple Regression

To fit a regression, it is important to identify some outliers and abnormal values. After checking each file, each max values of trip times are around 10800 except eighth file has a max value of 4294966. Each minimum of trip times are zeros except eighth file has a minimum of -6480. It is accordant with the boundary values of *total amount – tolls amount*. This might be some mistake of data itself. And since there are more than 173 million observations, the abnormal value can be ignored.

3.1.1. Estimate

The regression model is :

$$total\ amount - tolls\ amount = \alpha + \beta \times trip\ time + \varepsilon, \varepsilon \sim N(0, \sigma^2)$$

Or $y = \alpha + \beta x$

And the solution to the least square estimate of coefficient can be written as:

$$b = \frac{\sum x_i y_i - \frac{1}{n}\sum x_i \sum y_i}{\sum x_i^2 - \frac{1}{n}\sum x_i \sum x_i}$$

$$a = \bar{y} - b\bar{x}$$

According to the equation, there is no need to subtract all the data and compute the estimate. We can read each observation line, cumulate each x, y, square x, and xy, delete read line and read the next line. This way will not consume much memory and time to read/write files.

The estimate: b=2.06x10^{-5} a=14.52.

3.1.2. Different Approaches

Given the former discussion, it is very slow to use R functions to read csv files line bye line, so I only use C routines to read file and do cumulation job. And compare the computing time of parallel process and lapply.

Table 4 Computing Time

Parallel Process		
User	**System**	**Elapsed**
0.460	1.138	23.159
Lapply Process		
User	**System**	**Elapsed**
196.056	46.742	867.244

Parallel process is about 36 times faster than doing job one by one. However, the computing time of parallel process is very confusing. In theory, the user time of parallel process should be the sum computing time of twelve machines and elapsed time is how long we wait in front of the computer. So user time should be much bigger than elapsed time. But the fact is opposite! User time is only unbelievable 0.460. At first, I thought my codes might be wrong, but I got the same results of lapply process. And some other students also get the similar computing time and no one knows the reason. So, I can only attribute this strange computing time to the chaos of servers.

3.2. Multiple Regression (Two)

Surcharge is added as the second predictor into the model. According to the pre- check, each max values of surcharges are around 15 except eighth file has a max value of 854.5. Each minimum of trip times are zeros except eighth file file has a minimum of -19.5. As we discussed before, the abnormal value is ignored.

3.2.1. Estimate

The regression model is :

$$total\ amount - tolls\ amount = \alpha + \beta_1 \times trip\ time + \beta_2 \times surcharge + \varepsilon$$
$$\varepsilon \sim N(0, \sigma^2)$$

Or $y = \alpha + \beta_1 x_1 + \beta_2 x_2$

And the solution to the least square estimate of coefficient can be written as:

$$b_1 = \frac{(\sum x_2^2)(\sum x_1 y) - (\sum x_1 x_2)(\sum x_2 y)}{(\sum x_1^2)(\sum x_2^2) - (\sum x_1 x_2)^2}$$
$$b_2 = \frac{(\sum x_1^2)(\sum x_2 y) - (\sum x_1 x_2)(\sum x_1 y)}{(\sum x_1^2)(\sum x_2^2) - (\sum x_1 x_2)^2}$$
$$a = \overline{Y} - b_1 \overline{X}_1 - b_2 \overline{X}_2$$

(Source: http://faculty.cas.usf.edu/mbrannick/regression/Reg2IV.html)

Similar cumulation can be done. The estimate: a = 8.98 ,$b_1 = 5.09 \times 10^{-5}$, $b_2 = 17.2834$.

3.2.2. Parallel Approach

From former result, the parallel approach is much faster than lapply. So I only consider the parallel approach. The user time and elapsed time still have an unexpected relation, but it is similar to what I get in the simple regression case.

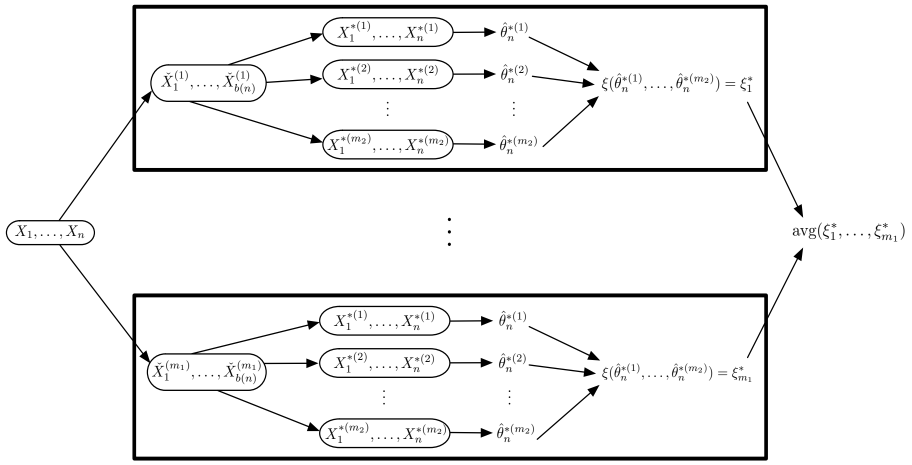
Table 5 Computing Time

Parallel Process		
User	**System**	**Elapsed**
0.493	1.259	29.469

3.3. Goodness of Estimate (Bag of Little Bootstraps)

3.3.1. Simplified Approach

As Michael talked in his Divide-and-Conquer and Statistical Inference for Big Data presentation, there are two sampling steps: (1) sample m_1 subsets of size b from a n population with no replacement;(2) resample n observations from each subset m_2 times with replacement.



$X_1, \ldots, X_n$
$\check{X}_1^{(1)}, \ldots, \check{X}_{b(n)}^{(1)}$
$X_1^{*(1)}, \ldots, X_n^{*(1)}$
$X_1^{*(2)}, \ldots, X_n^{*(2)}$
$X_1^{*(m_2)}, \ldots, X_n^{*(m_2)}$
$\hat{\theta}_n^{*(1)}$
$\hat{\theta}_n^{*(2)}$
$\hat{\theta}_n^{*(m_2)}$
$\xi(\hat{\theta}_n^{*(1)}, \ldots, \hat{\theta}_n^{*(m_2)}) = \xi_1^*$
$\check{X}_1^{(m_1)}, \ldots, \check{X}_{b(n)}^{(m_1)}$
$\xi(\hat{\theta}_n^{*(1)}, \ldots, \hat{\theta}_n^{*(m_2)}) = \xi_{m_1}^*$
$\mathrm{avg}(\xi_1^*, \ldots, \xi_{m_1}^*)$

Figure 2 Bag of Little Bootstraps (Michael I. Jordan)

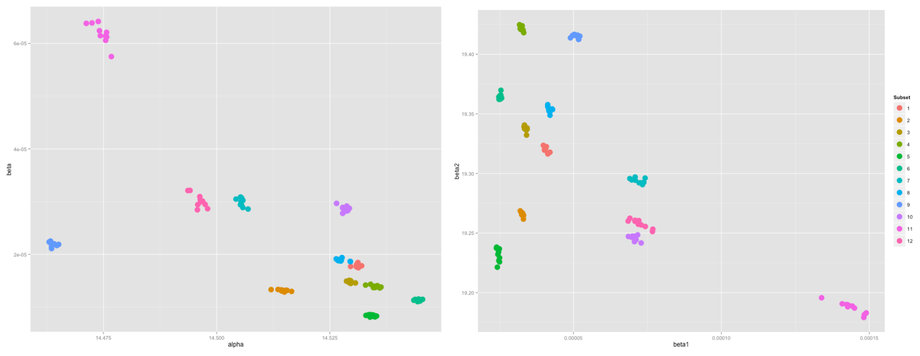
A straightforward approach would be getting sample index and obtained useful information from data set $m_1 \times m_2 \times n$ times. However reading pre-specified lines from such a big data set is really a pain, it is important to reduce how many times we read the data set.

Since in step(2), there are only b distinct values in each subsets. Instead reading data n times in each bootstrap data set, it is better to read data b times in each bootstrap data set and accumulate the data with corresponding weights(frequency). Thus we only read $m_1 \times m_2 \times b$ times. Although it is still a large number, it is much smaller than the straightforward one, in terms that b is often set to equal $n^{0.65}$.

3.3.2. Standard Errors of Estimate

In both simple regression and multiple regression cases, I set $m_1 = 12$, $m_2 = 10$, and $b = n^{0.65}$. (I could have tried a larger m_1 and m_2 but the sever was close to breaking down.)

I got 120 pairs of estimates for simple regression and multiple regression. In Figure 3, we can see there are naturally 12 clusters of estimates because of we have 12 sample subsets.



Simple Regression Multiple Regression

Figure 3 Estimates of Simple Regression(12 Subsets)

In Table 6 and Table 7,the standard errors are very small. However, the standard errors of b and b1 are quite big compare to their estimate value, which suggest they are not significant.

Table 6 Summaries of Simple Regression Estimates

	a	b
Mean	14.51539	2.257186e-05
Standard Error	0.02454764	1.385688e-05
T-statistics	591.3151	1.628928

Table 7 Summaries of Multiple Regression Estimates

	a	b1	b2
Mean	8.323389	5.340928e-05	19.30797
Standard Error	0.03263076	3.223961e-05	0.07076973
T-statistics	255.078	1.656635	272.8281

4. Conclusion

- Data Process Improvement

  If the data is big, then reading data into certain program object is time and memory consuming. Especially in R, it may needs more than three times of the memory size of the original data set to process into R data frame. It is important to refine the way to abstract useful information. In deciles and simple regression cases, the values are additive so we can read one line, process and delete. But it, like multiple regression (more than two predictors) is hard to break into additive pieces.
- Data Reading using R/C

  It is clear C is a faster way to loop and read lines than R. However, the program time for C is much slower than R. Even if I am equivalently skilled in both languages, the number of code lines using C is 69/16 times the number of that using R. In this case, the advantage in program time is erased by the computing time over such a big data.

  Meanwhile, memory allocation and type setting are important when call C from R. The maximum number of integer seems different when I pass a big integer vector and an integer number from R.
- Parallel Approach

  It is a great idea to do computation in parallel but it is pain to compete for the memory and computers

  in the crowded server. If 12 workers are set, the parallel approach will use more than 1/12 times of regular approach because masters need to time to allocate jobs and memory locate. When the server is busy, it spends more time to wait and get memory. In such way, parallel approach may not perform great as we expect.
- Bag of Little Bootstrap

It is expected that estimates from different subsets show obvious clusters. This may also suggest a bigger m_1 is needed to get a better average. It is also a trade-off among variance, bias, and computing time. The choice of m_2 is similar.

Appendix

1. Decils

1.1. Approach 1

R code:

```
#Percentile Using R readlines
rm(list = ls())
library(parallel)
breaks = 500
b = c(0, 587.79)
file2 = sapply(1:12, function(i) paste0("/home/data/NYCTaxis/trip_fare_", i,
".csv"))
y = c(10,11)
gethistR = function(f, bound, num, index){
 width = (bound[2]- bound[1])/num
 con = file(f , 'r')
 result = rep(0,num)
 line = readLines(con, n = 1)
 while( length( line <- readLines(con, n = 1)) > 0 ){
  spli = strsplit(line, ",")
  total.toll = as.numeric(spli[[1]][ index[2] ]) - as.numeric(spli[[1]][
index[1]])
  if(total.toll > bound[1] & total.toll < bound[2]){
   j = floor( (total.toll - bound[1])/width )
   result[j+1] = result[j+1]+1
  }
 }
 close(con)
 return(result)
}
cl = makeCluster(12, "FORK")
hist <- parLapplyLB(cl, 1:12, function(x) gethistR(file2[x], b, breaks, y) )
stopCluster(cl)
summ = hist[[1]]
for (i in c(2:12)) summ = summ + hist[[i]]
total = sum(summ, na.rm = TRUE)
cum = cumsum(summ)
decln = total*c(1:9)/10
```

```
decls = sapply(decln, function(i) which( i < cum)[1])
wid = (b[2]-b[1])/breaks
deciles = decls*wid + b[1]

save.image("histRonly.rda")
```

1.2. Approach 2

R code :

```
#newhist
#check minmax
#test simple regression on parallel
library(parallel)
setwd("/home/sxt999/run")
rm(list = ls())
cname = "hist"
compile = paste0("R CMD SHLIB ", cname,".c")
ccode = paste0(cname,".so")
system(compile)
y = c(10,11)
#x = 9
minmaxdif = function(c, f, index) {
  dyn.load(c)
  re = .C("minmaxdif", as.character(f), as.integer(index), as.numeric(0),
as.numeric(0), as.integer(0))
  dyn.unload(c)
  out = c( max = re[[3]], min = re[[4]], n = re[[5]] )
  return( out)
}
#file1 = sapply(1:12, function(i) paste0("/home/data/NYCTaxis/trip_data_",
i, ".csv"))
file2 = sapply(1:12, function(i) paste0("/home/data/NYCTaxis/trip_fare_", i,
".csv"))

cl = makeCluster(12, "FORK")
result <- parLapplyLB(cl, 1:12, function(x) minmaxdif(ccode, file2[x], y) )
stopCluster(cl)

b = c(median( unlist( lapply(result, "[[", 2))),
     median( unlist( lapply(result, "[[", 1))) )
breaks = 500

gethist = function(c ,f, bound, num, yindex){
  dyn.load(c)
  wid = (bound[2] - bound[1])/num
  re = .C("gethist", as.character(f), as.integer(yindex),
        as.integer( rep(0, num)), as.numeric(bound), as.numeric(wid))[[3]]
  class(re) = c( paste0("Width:", wid), paste0("Breaks:", num))
  dyn.unload(c)
  return(re)
}

cl = makeCluster(12, "FORK")
hist <- parLapplyLB(cl, 1:12, function(x) gethist(ccode, file2[x], b, breaks, y)
)
stopCluster(cl)
summ = hist[[1]]
for (i in c(2:12)) summ = summ + hist[[i]]
total = sum(summ)
cum = cumsum(summ)
decln = total*c(1:9)/10
decls = sapply(decln, function(i) which( i < cum)[1])
wid = (b[2]-b[1])/breaks
deciles = decls*wid + b[1]
save.image("hist.rda")
```

C code:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <R.h>

double getdif( const char *line, int num1, int num2){
 char *tok, *newline = strdup(line);
 char **bp = &newline;
 double dif;
 int i;
 for(i = 0; i<num2; i++) {
  tok = strsep(bp, ",");
  if( i == (num1-1) ) dif = -strtod(tok, NULL);
  if( i == (num2-1) ) dif = dif + strtod(tok, NULL);
 }
 return(dif);
}

void minmaxdif(char **file, int *index, double *max, double *min, int *n){
 FILE *stream = fopen(*file, "r");
 char line[1024];
 int i=0;
 while( fgets(line, 1024, stream))
 {
   if(i>0){
    char *tmp = strdup(line);
    double x;
    x = getdif(tmp, index[0], index[1]);
    if(x< *min) *min = x;
    if(x> *max) *max = x;
    free(tmp);
   };

   i++;
 }
 *n = i;
}

void gethist(char **file,int *target, int *result, double *bound, double
*width){
 FILE *stream = fopen(*file, "r");
 char line[1024];
 int i=0, j;
 double d;
 while( fgets(line, 1024, stream))
 {
   if(i>0){
    char *tmp = strdup(line);
    d = getdif(tmp, target[0], target[1]);
    if(d> bound[0]  && d< bound[1]){
    j = floor( (d-bound[0])/ (*width));
    result[j] = result[j] + 1;
    };
   free(tmp);
   };
   i++;
 }
}
```

1.3. Approach 3

R code:

```
#newhist
#check minmax
#test simple regression on parallel
setwd("/home/sxt999/run")
rm(list = ls())
cname = "hist"
compile = paste0("R CMD SHLIB ", cname,".c")
ccode = paste0(cname,".so")
system(compile)
y = c(10,11)
```

```
#x = 9
minmaxdif = function(c, f, index) {
  dyn.load(c)
  re = .C("minmaxdif", as.character(f), as.integer(index), as.numeric(0),
as.numeric(0), as.integer(0))
  dyn.unload(c)
  out = c( max = re[[3]], min = re[[4]], n = re[[5]] )
  return( out)
}
#file1 = sapply(1:12, function(i) paste0("/home/data/NYCTaxis/trip_data_",
i, ".csv"))
file2 = sapply(1:12, function(i) paste0("/home/data/NYCTaxis/trip_fare_", i,
".csv"))

t1 = system.time(result <- lapply(1:12, function(x) minmaxdif(ccode,
file2[x], y) ) )

b = c(median( unlist( lapply(result, "[[", 2))),
     median( unlist( lapply(result, "[[", 1))) )
breaks = 500

gethist = function(c ,f, bound, num, yindex){
  dyn.load(c)
  wid = (bound[2] - bound[1])/num
  re = .C("gethist", as.character(f), as.integer(yindex),
        as.integer( rep(0, num)), as.numeric(bound), as.numeric(wid))[[3]]
  class(re) = c( paste0("Width:", wid), paste0("Breaks:", num))
  dyn.unload(c)
  return(re)
}


t2 = system.time(hist <- lapply( 1:12, function(x) gethist(ccode, file2[x], b,
breaks, y) ) )
summ = hist[[1]]
for (i in c(2:12)) summ = summ + hist[[i]]
total = sum(summ)
cum = cumsum(summm)
decln = total*c(1:9)/10
decls = sapply(decln, function(i) which( i < cum)[1])
wid = (b[2]-b[1])/breaks
deciles = decls*wid + b[1]
#7.05348  8.22906  9.40464 10.58022 11.75580 14.10696 15.28254
18.80928 27.03834
save.image("histlapply.rda")
```

2. Regression

2.1. Check

R code:

```
#need to check medallion,hack_license,vendor_id
#the first three columns
library(parallel)
setwd("/home/sxt999/run")
rm(list = ls())
cname = "check"
compile = paste0("R CMD SHLIB ", cname,".c")
ccode = paste0(cname,".so")
system(compile)
check = function(c, f1, f2) {
  dyn.load(c)
  col1 = c(0,1,2,5)
  col2 = c(0,1,2,3)
  result = .C("check",as.character(f1),as.character(f2),as.integer(6),
     as.integer(4),as.integer(col1),as.integer(col2), as.integer(0))[[7]]
  dyn.unload(c)
  return(result)
}
file1 = sapply(1:12, function(i) paste0("/home/data/NYCTaxis/trip_data_", i,
".csv"))
file2 = sapply(1:12, function(i) paste0("/home/data/NYCTaxis/trip_fare_", i,
".csv"))

cl = makeCluster(12, "FORK")
t = system.time( result <- parLapplyLB(cl, 1:12, function(x) check(ccode,
file1[x], file2[x]) ) )
stopCluster(cl)
save.image("checkresult.rda")
```

C code:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <R.h>

void getfields( const char *line, int num, char **result){
 char *tok,*newline = strdup(line);
 char **bp = &newline;
 int i;
 for(i = 0; i<num; i++) {
   tok = strsep(bp, ",");
   result[i] = tok;
 }
}

void check(char **file1, char **file2, int *num1, int *num2, int *col1, int
*col2, int *result){
 FILE *stream1 = fopen(*file1, "r");
 FILE *stream2 = fopen(*file2,  "r");
 char line1[1024], line2[1024];
 int i=0,j;
 while( fgets(line1, 1024, stream1) && fgets(line2, 1024, stream2))
 {
    if(i>0){
     char *tmp1 = strdup(line1);
     char *tmp2 = strdup(line2);
     char **charp1[*num1], **charp2[*num2];

     getfields(tmp1, *num1, charp1);
     getfields(tmp2, *num2, charp2);
     for(j=0;j<*num1 && j <*num2;j++){
       if(strcmp(charp1[ col1[j] ],charp2[ col2[j] ]) !=0 ) *result = i;
     };
     free(tmp1);free(tmp2);
    };
    i++;
 }
}
```

2.2. Check Max & Min

R code:

```
#check minmax
library(parallel)
setwd("/home/sxt999/run")
rm(list = ls())
cname = "minmax"
compile = paste0("R CMD SHLIB ", cname,".c")
ccode = paste0(cname,".so")
system(compile)
y = c(10,11)
x1 = 9
x2 = 7
minmax = function(c, f, xindex) {
  dyn.load(c)
  re = .C("minmaxone", as.character(f), as.integer(xindex), as.numeric(0),
as.numeric(0), as.integer(0))
  dyn.unload(c)
  out = c( max = re[[3]] , min = re[[4]], n = re[[5]])
  return( out)
}
file1 = sapply(1:12, function(i) paste0("/home/data/NYCTaxis/trip_data_", i,
".csv"))
file2 = sapply(1:12, function(i) paste0("/home/data/NYCTaxis/trip_fare_", i,
".csv"))

cl = makeCluster(12, "FORK")
result1 <- parLapplyLB(cl, 1:12, function(x) minmax(ccode, file1[x], x1) )
result2 <- parLapplyLB(cl, 1:12, function(x) minmax(ccode, file2[x], x2))
stopCluster(cl)
save.image("minmax.rda")
```

C code:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <R.h>

double getfield( const char *line, int num){
 char *tok, *newline = strdup(line);
 char **bp = &newline;
 int i;
 for(i = 0; i<num; i++) tok = strsep(bp, ",");
 double target = strtod(tok, NULL);
 return(target);
}

void minmaxone(char **file, int *index, double *max, double *min, int *n ){
 FILE *stream = fopen(*file, "r");
 char line[1024];
 int i=0;
while( fgets(line, 1024, stream))
 {
   if(i>0){
    char *tmp = strdup(line);
    double x;
    x = getfield(tmp, *index);
    if(x< *min) *min = x;
    if(x> *max) *max = x;
    free(tmp);
   };

   i++;
 }
 *n = i;
}
```

2.3. Simple Regression

2.3.1. Parallel Approach

R code:

```
#test simple regression on parallel
library(parallel)
setwd("/home/sxt999/run")
rm(list = ls())
cname = "regsimp"
compile = paste0("R CMD SHLIB ", cname,".c")
ccode = paste0(cname,".so")
system(compile)
y = c(10,11)
x = 9
regsimp = function(c, f1, f2, yindex, xindex) {
  dyn.load(c)
  result = .C("reg",as.character(f1),as.character(f2),as.integer(yindex),
as.integer(xindex),
         as.numeric(rep(0,2)),as.integer(0), as.numeric(0),
as.numeric(0))
  out = result[[5]]
  out[3:5] = c(result[[7]], result[[8]],result[[6]])
  names(out) = c("x", "y","xy","x^2","n")
  dyn.unload(c)
  return(out)
}
file1 = sapply(1:12, function(i)
paste0("/home/data/NYCTaxis/trip_data_", i, ".csv"))
file2 = sapply(1:12, function(i)
paste0("/home/data/NYCTaxis/trip_fare_", i, ".csv"))

cl = makeCluster(12, "FORK")
t = system.time(
  result <- parLapplyLB(cl, 1:12, function(i) regsimp(ccode, file1[i],
file2[i], y, x) ) )
stopCluster(cl)
summ = sapply(1:5 , function(i) sum(unlist(lapply(result,"[[",i ))))

beta = (summ[3]- summ[1]*summ[2]/summ[5]) / ( summ[4] -
(summ[1])*(summ[1])/summ[5])

alpha = summ[2]/summ[5] - beta * summ[1]/summ[5]
```

```
save.image("result.rda")
```

C code:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <R.h>

double getfield( const char *line, int num){
 char *tok, *newline = strdup(line);
 char **bp = &newline;
 int i;
 for(i = 0; i<num; i++) tok = strsep(bp, ",");
 double target = strtod(tok, NULL);
 return(target);
}
double getdif( const char *line, int num1, int num2){
 char *tok, *newline = strdup(line);
 char **bp = &newline;
 double dif;
 int i;
 for(i = 0; i<num2; i++) {
  tok = strsep(bp, ",");
  if( i == (num1-1) ) dif = -strtod(tok, NULL);
  if( i == (num2-1) ) dif = dif + strtod(tok, NULL);
 }
 return(dif);
}

void reg(char **file1, char **file2, int *yindex, int *xindex, double
*sum,int *n,
double *inter, double *square ){
 FILE *stream1 = fopen(*file1, "r");
 FILE *stream2 = fopen(*file2,  "r");
 char line1[1024], line2[1024];
 int i=0;
while( fgets(line1, 1024, stream1) && fgets(line2, 1024, stream2))
 {
   if(i>0){
    char *tmp1 = strdup(line1), *tmp2 = strdup(line2);
    double x,y;
    y = getdif(tmp2, yindex[0],yindex[1]);
    x = getfield(tmp1, xindex[0]);
    sum[0] = sum[0] + x;
    sum[1] = sum[1] + y;
```

```
     *inter = *inter + x*y;
     *square = *square + x*x;
     free(tmp1);free(tmp2);
    };
    i++;
  }
  *n = i;
 }
```

2.3.2. Lapply Approach

R code:

```
#SimpReg lapply
#test simple regression on parallel
setwd("/home/sxt999/run")
rm(list = ls())
cname = "regsimp"
compile = paste0("R CMD SHLIB ", cname,".c")
ccode = paste0(cname,".so")
system(compile)
y = c(10,11)
x = 9
regsimp = function(c, f1, f2, yindex, xindex) {
  dyn.load(c)
  sum = rep(0,4)
  result = .C("reg",as.character(f1),as.character(f2),as.integer(yindex),
as.integer(xindex),
              as.numeric(sum),as.integer(0), as.numeric(0), as.numeric(0))
  out = result[[5]]
  out[3:5] = c(result[[7]], result[[8]],result[[6]])
  names(out) = c("x", "y","xy","x^2","n")
  dyn.unload(c)
  return(out)
}
file1 = sapply(1:12, function(i)
paste0("/home/data/NYCTaxis/trip_data_", i, ".csv"))
file2 = sapply(1:12, function(i)
paste0("/home/data/NYCTaxis/trip_fare_", i, ".csv"))

t = system.time(
  result <-lapply(1:12, function(i) regsimp(ccode, file1[i], file2[i], y, x) ) )

summ = sapply(1:5 , function(i) sum(unlist(lapply(result,"[[",i ))))

beta = (summ[3]- summ[1]*summ[2]/summ[5]) / ( summ[4] -
(summ[1])*(summ[1])/summ[5])
```

```
alpha = summ[2]/summ[5] - beta * summ[1]/summ[5]
save.image("resultLapply.rda")
```

2.4. Multiple Regression

R code

```
#test two regression on parallel
library(parallel)
rm(list = ls())
setwd("/home/sxt999/run")
cname = "regtwo"
compile = paste0("R CMD SHLIB ", cname,".c")
ccode = paste0(cname,".so")
system(compile)
y = c(10,11)
x = c(9,7)
regtwo = function(c, f1, f2, yindex, xindex) {
  dyn.load(c)
  #yindex = c(10,11)
  #xindex = c(9,7)
  result = .C("regtwo",as.character(f1),as.character(f2),as.integer(yindex),
as.integer(xindex),
            as.numeric(rep(0,3)), as.numeric(rep(0,2)), as.numeric(0),
as.numeric(rep(0,2)), as.integer(0))
  out = result[[5]]
  out[4:5] = result[[6]]
  out[6] = result[[7]]
  out[7:8] = result[[8]]
  out[9] = result[[9]]
  names(out) = c("x1", "x2", "y","x1^2", "x2^2","x1*x2","x1*y","x2*y","n")
  dyn.unload(c)
  return(out)
}
file1 = sapply(1:12, function(i) paste0("/home/data/NYCTaxis/trip_data_", i,
".csv"))
file2 = sapply(1:12, function(i) paste0("/home/data/NYCTaxis/trip_fare_", i,
".csv"))

cl = makeCluster(12, "FORK")
t = system.time(
  result <- parLapplyLB(cl, 1:12, function(i) regtwo(ccode, file1[i], file2[i], y,
x) ) )
stopCluster(cl)

summ = sapply(1:9 , function(i) sum(unlist(lapply(result,"[[",i ))))
```

```
beta1 = (summ[5]*summ[7] - summ[6]*summ[8])/(summ[4]*summ[5] -
summ[6]^2 )
beta2 = (summ[4]*summ[8] - summ[6]*summ[7])/(summ[4]*summ[5] -
summ[6]^2 )
alpha = summ[3]/summ[9] - beta1 * summ[1]/summ[9] - beta2 *
summ[2]/summ[9]

save.image("resulttwo.rda")
```

C code

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <R.h>

double getfield( const char *line, int num){
 char *tok, *newline = strdup(line);
 char **bp = &newline;
 int i;
 for(i = 0; i<num; i++) tok = strsep(bp, ",");
 double target = strtod(tok, NULL);
 return(target);
}
double getdif( const char *line, int num1, int num2){
 char *tok, *newline = strdup(line);
 char **bp = &newline;
 double dif;
 int i;
 for(i = 0; i<num2; i++) {
  tok = strsep(bp, ",");
  if( i == (num1-1) ) dif = -strtod(tok, NULL);
  if( i == (num2-1) ) dif = dif + strtod(tok, NULL);
 }
 return(dif);
}

void regtwo(char **file1, char **file2, int *yindex, int *xindex ,double *sum,
  double *square, double *interx, double *interxy, int *n){
 FILE *stream1 = fopen(*file1, "r");
 FILE *stream2 = fopen(*file2,  "r");
 char line1[1024], line2[1024];
 int i=0;
while( fgets(line1, 1024, stream1) && fgets(line2, 1024, stream2))
 {
   if(i>0){
    char *tmp1 = strdup(line1), *tmp2 = strdup(line2);
    double x1,x2,y;
    y = getdif(tmp2, yindex[0],yindex[1]);
    x1 = getfield(tmp1, xindex[0]);
    x2 = getfield(tmp2, xindex[1]);
    sum[0] = sum[0] + x1;
    sum[1] = sum[1] + x2;
    sum[2] = sum[2] + y;
    square[0] = square[0] + x1*x1;
    square[1] = square[1] + x2*x2;
    interx[0] = interx[0] + x1*x2;
    interxy[0] = interxy[0] + x1*y;
    interxy[1] = interxy[1] + x2*y;
    free(tmp1);free(tmp2);
   };

   i++;

 }
 *n = i;
}
```

3. **BLB**

   3.1. **Simple Regression**

   R code

```
#bag of little bootstrap
library(parallel)
setwd("/home/sxt999/run")
rm(list = ls())
cname = "regsimpblb"
compile = paste0("R CMD SHLIB ", cname,".c")
ccode = paste0(cname,".so")
system(compile)

regsamp = function(c, f1, f2, yindex, xindex, samp, weight){
  dyn.load(c)
  re = .C("regsamp", as.character(f1), as.character(f2), as.integer(yindex),
as.integer(xindex),
        as.numeric( rep(0,4)), as.integer(0),as.integer(samp),
as.integer(weight), as.integer(0) )
  dyn.unload(c)
  out = re[[5]]
  out[5] = re[[9]]
  out[6] = re[[6]]
  names(out) = c("x", "y","xy","x^2","stop","n")
  return(out)
}
```

```
y= c(10,11)
x = 9
file1 = sapply(1:12, function(i) paste0("/home/data/NYCTaxis/trip_data_", i,
".csv"))
file2 = sapply(1:12, function(i) paste0("/home/data/NYCTaxis/trip_fare_", i,
".csv"))
ns =
c(14776616,13990177,15749229,15100469,15285050,14385457,1382384
1,12597110,14107694,15004557,
      14388452,13971119 )
boot = function(c,fs1, fs2, bindex.sort, n, yindex, xindex, seed ){
  set.seed(seed)
  boot = sample(bindex.sort, n, replace = TRUE)
  count.boot = table(boot)
  names.boot = as.integer(names(count.boot))
  summ = rep(0, length(bindex.sort))
  summ[match(names.boot, bindex.sort)] = count.boot
  #summ is the corresponding frequency of bindex.sort

  sampindex = bindex.sort
  w = summ

  result = as.list( 1:12)
  for(i in c(1:12)){
    result[[i]] = regsamp(c, fs1[i], fs2[i], yindex, xindex, sampindex, w)
    sampindex = sampindex[ (result[[i]][5]+1) : length(sampindex)] -
result[[i]][6]
    w = w[  (result[[i]][5] +1) : length(w) ]
  }
  boot.summ = sapply(1:4 , function(i) sum(unlist(lapply(result,"[[",i ))))
  beta = (boot.summ[3]- boot.summ[1]*boot.summ[2]/n) /
    ( boot.summ[4] - (boot.summ[1])*(boot.summ[1])/n)

  alpha = boot.summ[2]/n - beta * boot.summ[1]/n
  return( c(alpha, beta))
}
lbl = function(c, fs1, fs2, yindex, xindex, n.s, seed, power, m){
  set.seed(seed)
  n = sum(n.s)
  b = n^power
  bindex = sample(1:n, b, replace = FALSE)
  bindex.sort = bindex[order(bindex)]
  estimate = lapply(1:m, function(i) boot(c,fs1,fs2, bindex.sort, n, yindex,
xindex,i ))
  return(estimate)
}
```

```
cl = makeCluster(12, "FORK")
t = system.time(
  result <- parLapplyLB(cl, 12:23, function(i) lbl(ccode, file1, file2, y, x, ns, i,
0.65, 10) ) )
stopCluster(cl)

save.image("resultBLB.rda")
```

C code

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <R.h>

double getfield( const char *line, int num){
 char *tok, *newline = strdup(line);
 char **bp = &newline;
 int i;
 for(i = 0; i<num; i++) tok = strsep(bp, ",");
 double target = strtod(tok, NULL);
 return(target);
}
double getdif( const char *line, int num1, int num2){
 char *tok, *newline = strdup(line);
 char **bp = &newline;
 double dif;
 int i;
 for(i = 0; i<num2; i++) {
  tok = strsep(bp, ",");
  if( i == (num1-1) ) dif = -strtod(tok, NULL);
  if( i == (num2-1) ) dif = dif + strtod(tok, NULL);
 }
 return(dif);
}

void regsamp(char **file1, char **file2, int *yindex, int *xindex, double
*sum,int *n, int *samp, int *weight, int *stop ){
 FILE *stream1 = fopen(*file1, "r");
 FILE *stream2 = fopen(*file2,  "r");
 char line1[1024], line2[1024];
 int i=0,j=0;
while( fgets(line1, 1024, stream1) && fgets(line2, 1024, stream2))
 {
   if(i>0 && i == samp[ j ]){
    char *tmp1 = strdup(line1), *tmp2 = strdup(line2);
    double x,y;
    y = getdif(tmp2, yindex[0],yindex[1]);
    x = getfield(tmp1, *xindex);
    sum[0] = sum[0] + x*weight[j];
    sum[1] = sum[1] + y*weight[j];
    sum[2] = sum[2] + x*y*weight[j];
    sum[3] = sum[3] + x*x*weight[j];
    free(tmp1);free(tmp2);
    j++;
   };

   i++;

 }
 *n = i;
 *stop = j;
}
```

3.2. Multiple Regression

R code

```
#reg two blb
#bag of little bootstrap
library(parallel)
setwd("/home/sxt999/run")
rm(list = ls())
cname = "regtwoblb"
compile = paste0("R CMD SHLIB ", cname,".c")
ccode = paste0(cname,".so")
system(compile)

regtwosamp = function(c, f1, f2, yindex, xindex, samp, weight) {
  dyn.load(c)
  #yindex = c(10,11)
  #xindex = c(9,7)
  result =
.C("regtwosamp",as.character(f1),as.character(f2),as.integer(yindex),
as.integer(xindex),
          as.numeric(rep(0,3)), as.numeric(rep(0,2)), as.numeric(0),
as.numeric(rep(0,2)),
          as.integer(samp), as.integer(weight), as.integer(0), as.integer(0))
  out = result[[5]]
  out[4:5] = result[[6]]
  out[6] = result[[7]]
  out[7:8] = result[[8]]
  out[9] = result[[11]]
  out[10] = result[[12]]
  names(out) = c("x1", "x2", "y","x1^2", "x2^2","x1*x2","x1*y","x2*y","stop",
"n")
  dyn.unload(c)
  return(out)
}

y = c(10,11)
x = c(9,7)
file1 = sapply(1:12, function(i) paste0("/home/data/NYCTaxis/trip_data_", i,
".csv"))
file2 = sapply(1:12, function(i) paste0("/home/data/NYCTaxis/trip_fare_", i,
".csv"))
ns =
c(14776616,13990177,15749229,15100469,15285050,14385457,1382384
1,12597110,14107694,15004557,
       14388452,13971119 )

boot = function(c, fs1,fs2, bindex.sort, n, yindex, xindex, seed ){
  set.seed(seed)
  boot = sample(bindex.sort, n, replace = TRUE)
  count.boot = table(boot)
  names.boot = as.integer(names(count.boot))
  summ = rep(0, length(bindex.sort))
  summ[match(names.boot, bindex.sort)] = count.boot
  #summ is the corresponding frequency of bindex.sort

  sampindex = bindex.sort
  w = summ

  result = as.list( 1:12)
  for(i in c(1:12)){
    result[[i]] = regtwosamp(c, fs1[i], fs2[i], yindex, xindex, sampindex, w)
    sampindex = sampindex[ (result[[i]][9]+1) : length(sampindex)] -
result[[i]][10]
    w = w[  (result[[i]][9] +1) : length(w) ]
  }
  boot.summ = sapply(1:8 , function(i) sum(unlist(lapply(result,"[[",i ))))

  beta1 = (boot.summ[5]*boot.summ[7] - boot.summ[6]*boot.summ[8])/
    (boot.summ[4]*boot.summ[5] - boot.summ[6]^2 )

  beta2 = (boot.summ[4]*boot.summ[8] - boot.summ[6]*boot.summ[7])/
    (boot.summ[4]*boot.summ[5] - boot.summ[6]^2 )
  alpha = boot.summ[3]/n - beta1 * boot.summ[1]/n - beta2 *
boot.summ[2]/n
```

```
  return( c(alpha, beta1, beta2))
}

lbl = function(c, fs1, fs2, yindex, xindex, n.s, seed, power, m){
  set.seed(seed)
  n = sum(n.s)
  b = n^power
  bindex = sample(1:n, b, replace = FALSE)
  bindex.sort = bindex[order(bindex)]
  estimate = lapply(1:m, function(i) boot(c, fs1,fs2, bindex.sort, n, yindex,
xindex,i ))
  return(estimate)

}

cl = makeCluster(12, "FORK")
t = system.time(
  result <- parLapplyLB(cl, 12:23, function(i) lbl(ccode, file1, file2, y, x, ns, i,
0.65, 10) ) )
stopCluster(cl)

save.image("resulttwoBLB.rda")
```

C code

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <R.h>

double getfield( const char *line, int num){
 char *tok, *newline = strdup(line);
 char **bp = &newline;
 int i;
 for(i = 0; i<num; i++) tok = strsep(bp, ",");
 double target = strtod(tok, NULL);
 return(target);
}
double getdif( const char *line, int num1, int num2){
 char *tok, *newline = strdup(line);
 char **bp = &newline;
 double dif;
 int i;
 for(i = 0; i<num2; i++) {
  tok = strsep(bp, ",");
  if( i == (num1-1) ) dif = -strtod(tok, NULL);
  if( i == (num2-1) ) dif = dif + strtod(tok, NULL);
 }
 return(dif);
}

void regtwosamp(char **file1, char **file2, int *yindex, int *xindex ,double
*sum,
  double *square, double *interx, double *interxy,
 int *samp, int *weight, int *stop, int *n){
 FILE *stream1 = fopen(*file1, "r");
 FILE *stream2 = fopen(*file2,  "r");
 char line1[1024], line2[1024];
 int i=0,j=0;
while( fgets(line1, 1024, stream1) && fgets(line2, 1024, stream2))
 {
   if(i>0 && i == samp[j] ){
    char *tmp1 = strdup(line1), *tmp2 = strdup(line2);
    double x1,x2,y;
    y = getdif(tmp2, yindex[0],yindex[1]);
    x1 = getfield(tmp1, xindex[0]);
    x2 = getfield(tmp2, xindex[1]);
    sum[0] = sum[0] + x1*weight[j];
    sum[1] = sum[1] + x2*weight[j];
    sum[2] = sum[2] + y*weight[j];
    square[0] = square[0] + x1*x1*weight[j];
    square[1] = square[1] + x2*x2*weight[j];
    interx[0] = interx[0] + x1*x2*weight[j];
    interxy[0] = interxy[0] + x1*y*weight[j];
    interxy[1] = interxy[1] + x2*y*weight[j];
    free(tmp1);free(tmp2);
    j++;
   };

   i++;
 }
 *stop = j;
 *n = i;
}
```