1 Bag of Little Bootstraps

- Program codes:

R Code

```
###################
#BLB_lin_reg_job.R#
###################

mini <- FALSE

# The part to generate sim_num is omitted.

# Find r and s indices:
s_index<-ceiling((sim_num-1000)/50)
r_index<-(sim_num-1000)%%50
if (r_index==0) r_index<-50

# Load packages:
library(BH)
library(bigmemory.sri)
library(bigmemory)
library(biganalytics)

# I/O specifications:
datapath <- "/home/pdbaines/data"
outpath <- "output/"

# mini or full?
if (mini){
  rootfilename <- "blb_lin_reg_mini"
} else {
  rootfilename <- "blb_lin_reg_data"
}

# Read in the data
descriptorfilename <- paste0(rootfilename,".desc")
descriptorfile <- paste(datapath,descriptorfilename,sep="/")
dat<-attach.big.matrix(dget(descriptorfile),backingpath=datapath)
```

```
# Find sub-sample size b
n<-nrow(dat)
gamma<-0.7
b<-floor(n^gamma)

# Sample b rows from the full dataset
index<-{set.seed(s_index); sample(1:n, b, replace=F)}
subsample<-as.data.frame(dat[index,])
# Draw a bootstrap sample of size n from the subsample
rep.times<-{set.seed(sim_num);rmultinom(1,n, prob=rep(1, b)/b)}

# Fit a weighted least squares regression
if (mini){
  fit<-lm(subsample$V41~.-1, data=subsample, weights=rep.times)
} else {
  fit<-lm(subsample$V1001~.-1, data=subsample, weights=rep.times)
}
coef<-fit$coef

# Save estimates to file
outfile=paste0("output/", "coef_", sprintf("%02d", s_index), "_",
              sprintf("%02d", r_index), ".txt")
write.table(coef, file=outfile, sep=",", row.names=F, col.names=F)
```

R Code

- The index plot is shown in Figure 1 in the appendix.
- Based on the index plot, the BLB SE estimates fluctuate around 0.01. Since the SE's should be around 0.01 for this example, the algorithm works well in this case.

2 MapReduce via Hadoop

- The 2D histogram produced from my MapReduce results is shown in Figure 2 in the appendix.
- In my map function, firstly I read in the data from the standard input and obtain pairs of (x,y) values. Then I extract the value of x and y for each pair and compute the lower bounds for the bin containing the pair. The lower bounds for x and y are combined and

used as the key in the output, accompanied with 1 as the corresponding value. Then this output is used as the input for my reduce function, where I sum the count for all the pairs with the same key and output both the lower bounds and upper bounds for x and y and the corresponding cumulative counts in csv format. It is important to make sure that the bounds and the counts are computed correctly and the output is in a decent format for easy manipulation later on. The details can be found in the attached codes. My program seemed to be reasonably quick and it took around 10 minutes to run it with 1 large master node and 2 large compute nodes. Basically I followed the codes Prof. Baines provided in the lecture and I think one possible improvement is to find more efficient ways to compute the bounds for the bin.

My Map function is:

Map Function

```
#!/usr/bin/env python

import sys

# input comes from STDIN (standard input)
for line in sys.stdin:
    line = line.strip()
    pairs = line.split('\n')
    # extract x and y coordinates
    for pair in pairs:
        pair = pair.strip()
        pair = pair.split()
        x = float(pair[0])
        y = float(pair[1])

        # find the lower bounds of the bin
        if round(x, 1) < x:
            x_low = round(x, 1)
        elif round(x, 1) > x:
            x_low = round(x, 1) - 0.1
        else:
            x_low = x - 0.1

        if round(y, 1) < y:
            y_low = round(y, 1)
        elif round(y, 1) > y:
            y_low = round(y, 1) - 0.1
        else:
            y_low = y - 0.1

        # combine the lower bounds of both x and y
        xy_low = str(x_low) + str('_') + str(y_low)

        print '%s\t%s' % (xy_low, 1)
```

Map Function

My Reduce function is:

Reduce Function

```
#!/usr/bin/env python

from operator import itemgetter
import sys

current_xy_low = None
current_count = 0
xy_low = None

# input comes from STDIN
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # parse the input we got from mapper.py
    xy_low, count = line.split('\t', 1)

    # convert count (currently a string) to int
    try:
        count = int(count)
    except ValueError:
        # count was not a number, so silently
        # ignore/discard this line
        continue

    # if the key is the same, sum the counts
    if current_xy_low == xy_low:
        current_count += count
    else:
        if current_xy_low:
            # write result to STDOUT
            x_low, y_low = current_xy_low.split('_', 1)
            out_str = x_low + ',' + str(float(x_low)+0.1) + ',' + y_low +
            ',' + str(float(y_low)+0.1)
            print '%s,%s' % (out_str, current_count)
        current_count = count
        current_xy_low = xy_low

# output the last word if needed
if current_xy_low == xy_low:
    x_low, y_low = current_xy_low.split('_', 1)
    out_str = x_low + ',' + str(float(x_low)+0.1) + ',' + y_low +
    ',' + str(float(y_low)+0.1)
    print '%s,%s' % (out_str, current_count)
```

Reduce Function

3 Data Manipulation with Hive

- The scatterplot is shown in Figure 3 in the appendix. The within-group means seem to center around zero based on the plot.
- The hive commands:

Hive Commands

```
# log into hive
hive

# create a table which contains two columns which are separated by tab
CREATE TABLE gp (
 Group INT,
 Value FLOAT
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';

# load the data (which has been transferred to hadoop local using
# copyToLoal) into table gp
```

```
LOAD DATA LOCAL INPATH 'groups.txt' OVERWRITE INTO TABLE gp;

# compute the within-group means and save them in the directory res_mean
INSERT OVERWRITE DIRECTORY 'res_mean' SELECT avg(gp.Value)
FROM gp
GROUP BY gp.Group;

# compute the within-group variances and save them in the directory res_var
INSERT OVERWRITE DIRECTORY 'res_var' SELECT variance(gp.Value)
FROM gp
GROUP BY gp.Group;

# use Ctrl-C to quit hive
```

Hive Commands

A Plots

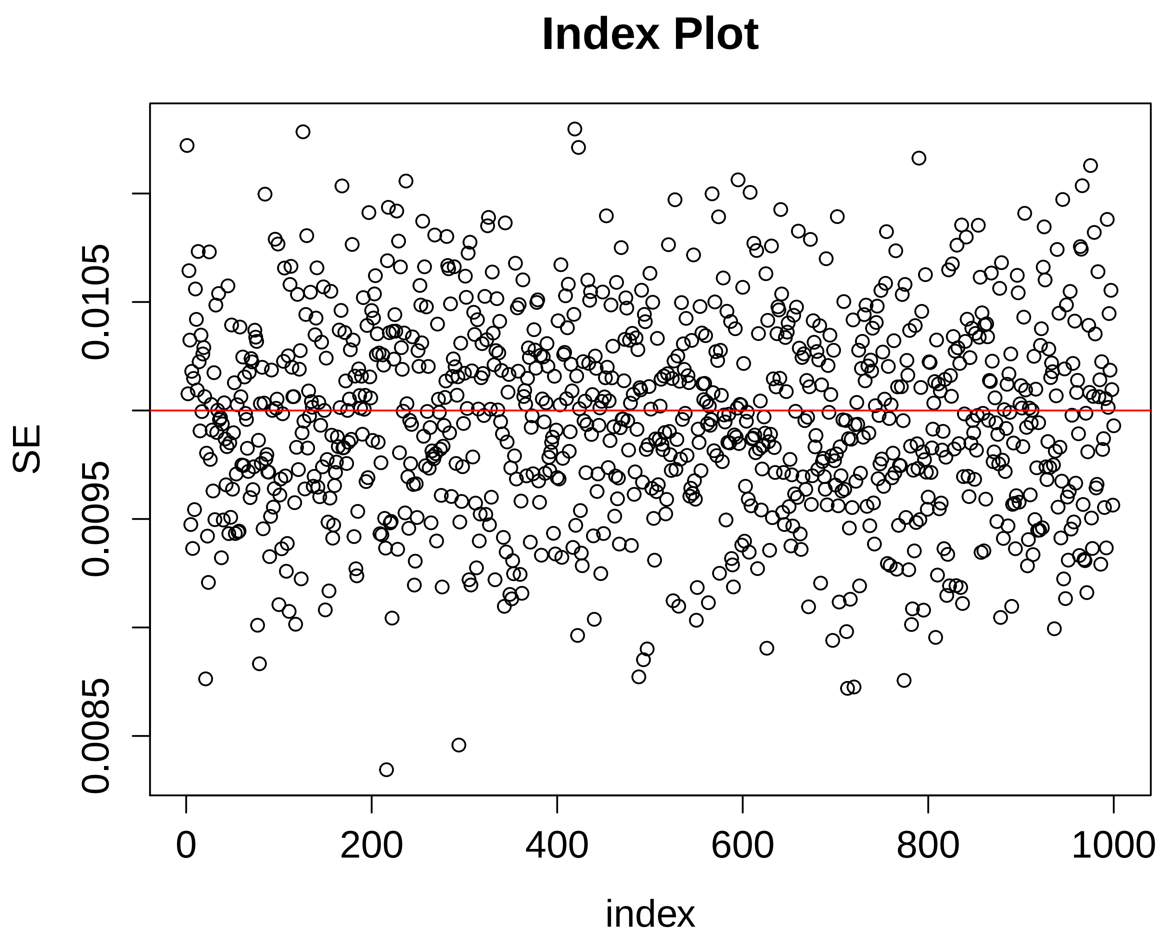

Figure 1: Index plot of the SE values obtained in part (f)

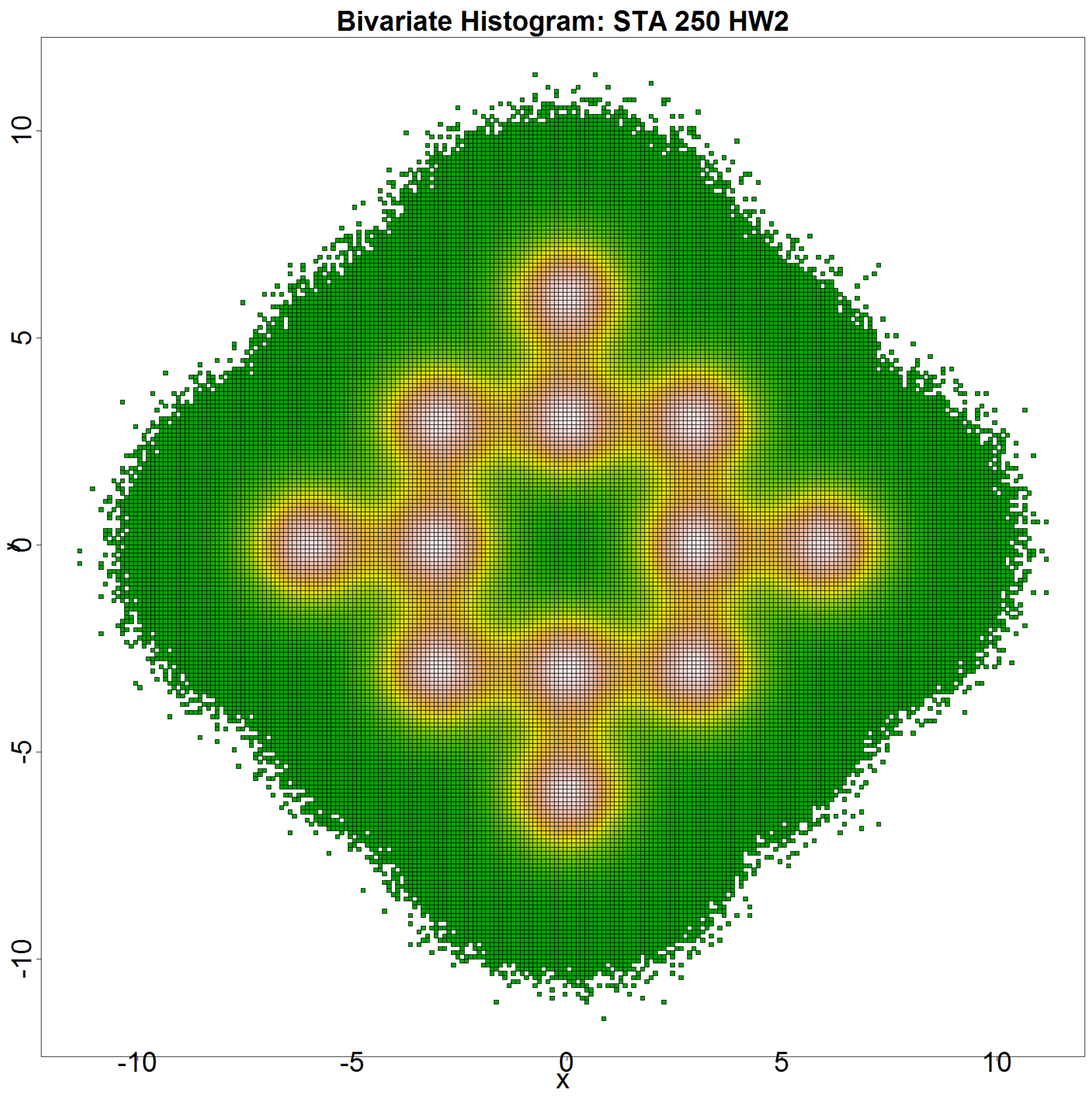

Figure 2: Histogram of simulated mean response.

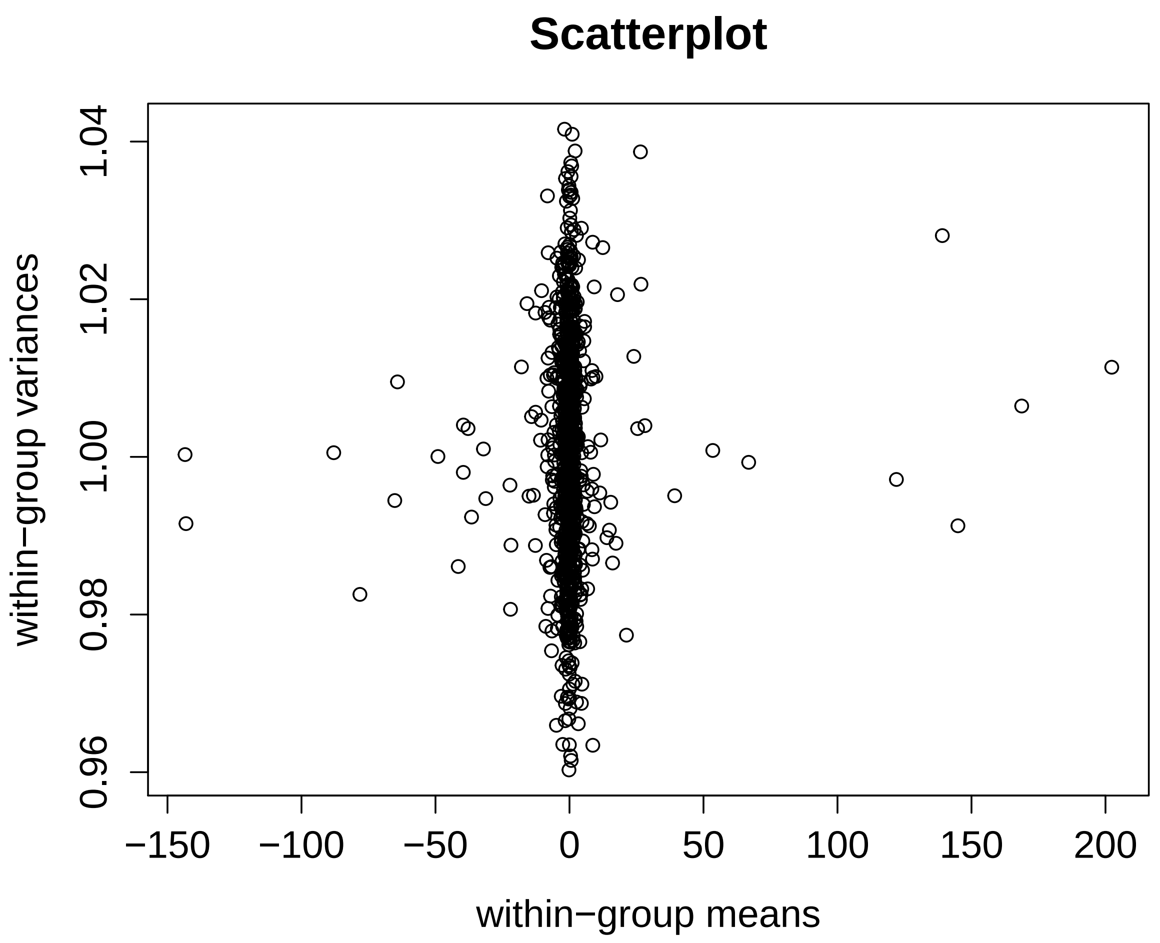

Figure 3: Scatterplot of the within-group means vs. the within-group variances.