Vladimir Pchelin
Last assignment

git@bitbucket.org:ph7vov/sta242.git

Goals. In this assignment I do the following:

- compute the deciles of the total amount minus the tolls
- fit a linear regression predicting total amount less the tolls using trip time as the predictor
- get errors for coefficients of this fit
- use multiple regression and add the surcharge as a regressor
- get errors for coefficients of the multiple regression

Tools I used.

- to process the `.csv` files I used basic `shell` commands
- for algebraic operations I used mostly `awk`
- for random selection I employed `perl`
- more complicated operations with data were done in `C++`
- `R` to create plots and analyze random sample

My strategy and what I did. In this paragraph, I list all my operations with the data in a chronological order. First, I `cut` the three columns we were interested in (”surcharge”,”time”,”total-tolls”) into three different files. My computer spent the major part of the overall time on this operation. The subtraction was performed with `awk`. I also converted ”surcharge” and ”total-tolls” into cents. This was quite useful because all the numbers became integers. *That's why I didn't have any problems with numerical imprecision.* For convenience I also concatenated each 12 files into single ones. After that, using `C+++`, I obtained the distribution of ”total-tolls” via the ”bin” method. Also, I got all the necessary vector products for the linear regression in `C+++`. Applying `paste` and `perl -ne 'print if (rand() < .001)'` I got my random sample. This sample was then processed in `R`.

The deciles. The next figure shows what I obtained using `C+++` and the ”bin” strategy. This is a smoothed relative true distribution, ”true” means that the initial discrete distribution contains all the available information from our data.

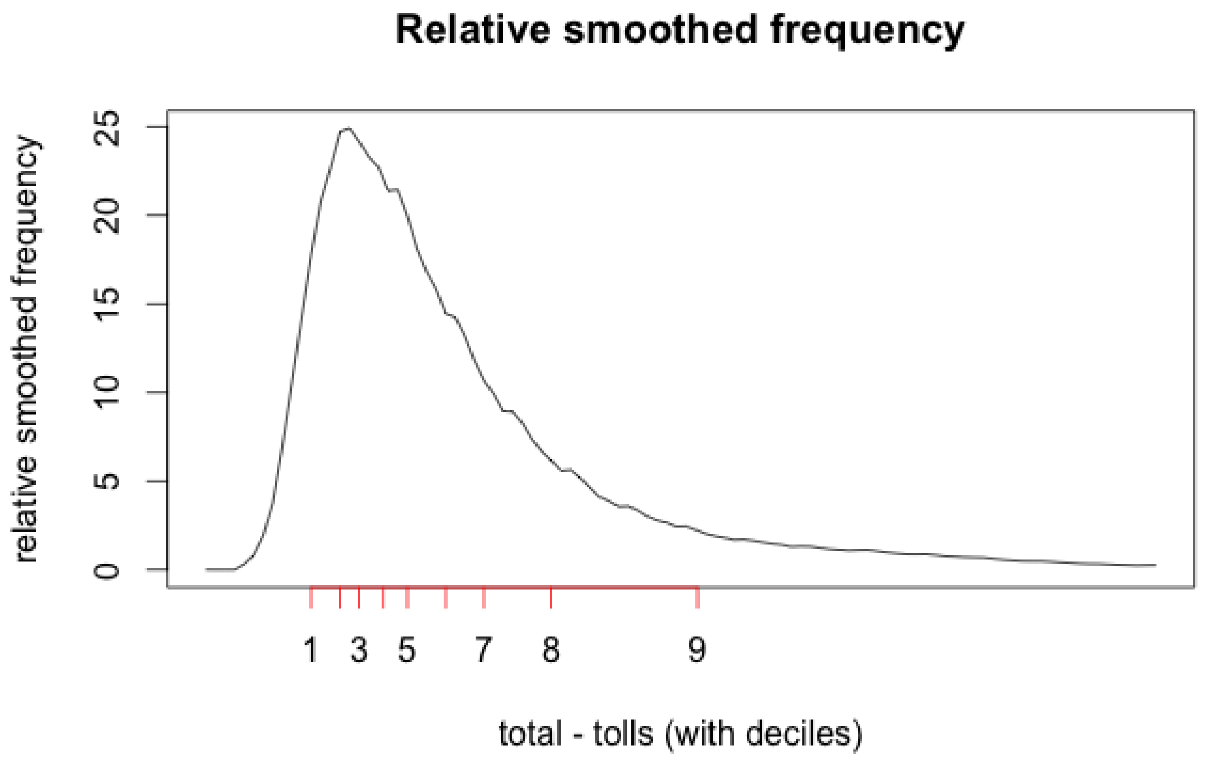

Figure 1

The graph above strongly depends on how we smooth. I averaged the the initial discrete distribution with the precision ± 50 cents; the frequencies on this plot are only *relative*. On the x-axis there are marks that represent 9 deciles. These deciles are (in cents)

decile	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
value	600	750	850	975	1100	1300	1500	1850	2612

The next figure is obtained from the sampling.

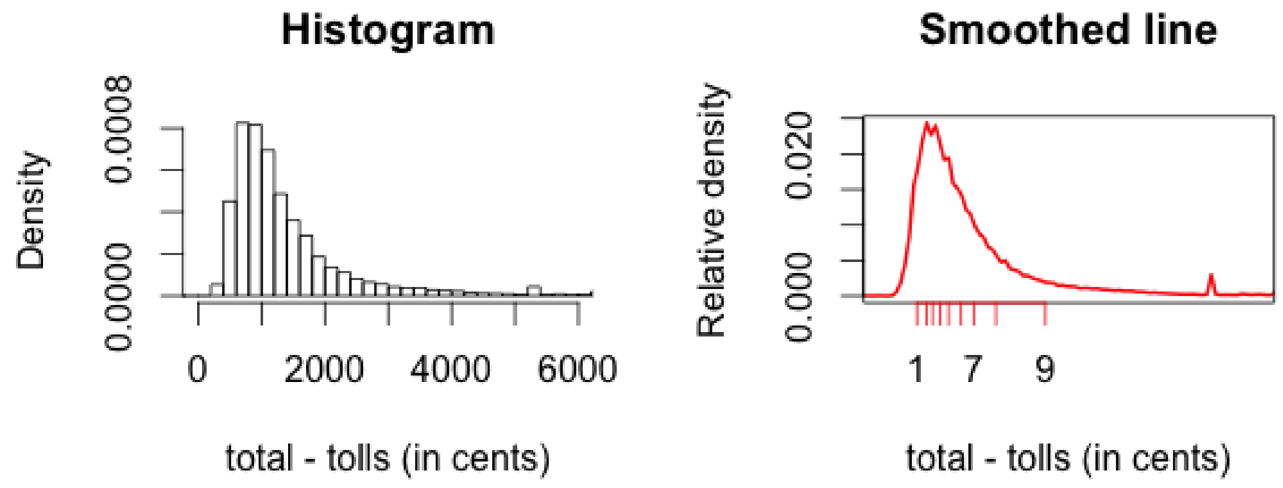

Figure 2

The deciles here are

decile	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
value	600	750	850	975	1100	1300	1500	1850	2630

Let's compare the results. First of all, the plots, as well as, the decile values are almost exactly the same. Nevertheless, the following should be mentioned. To compute deciles with `C++` I used my own function, while for the sample I used `quantiles()`. These two functions might be different in handling discontinuities. That's, probably, the reason why I have "2612" and "2630". Plot methods and smoothing were also different. Unfortunately, I could not apply the same plot methods in both case because of performance issues.

The decile values mainly depend on how we treat discontinuities. The latter affects the computed values much more than the precision of our computation. The precision of the computation is higher than 10 cents since the density in a range ± 10 is much larger than zero.

Simple regression and multiple regression. To get linear regression in `C++` I just computed the entires of the corresponding $X'X$ matrix and $X'Y$ vector. First of all, I noticed that there were huge outliers in "time" data; I dropped these outliers. In this paragraph, I use 1 to indicate that the correspondence to the simple regression and 2 to indicate that the correspondence to the regression with two predictors. Then the computed coefficients $\beta_0, \beta_{time}, \beta_{surcharge}$ are

```
> solve(XX1)%*%XY1
[1,] 244.081450
[2,]   1.604833
```

and

```
 > solve(XX2)%*%XY2
 [1,] 232.248080
 [2,]   1.606063
 [3,]   0.341510
```

The coefficients $\beta_0, \beta_{time}, \beta_{surcharge}$ correspond to $1, 2, 3$ rows. Now let's get the covariance matrix for the coefficients:

```
> solve(XX1)*MSE1
               [,1]            [,2]
[1,]  0.02072970749 -0.00001767336462
[2,] -0.00001767336  0.00000002345008
```

for the first model and for the second model

```
> solve(XX2)*MSE2
               [,1]             [,2]             [,3]
[1,]  0.02784667639 -0.00001841334481 -0.00020539508637
[2,] -0.00001841334  0.00000002352702  0.00000002135576
[3,] -0.00020539509  0.00000002135576  0.00000592768394
```

Easy to see that all the coefficients are highly significant. I don't perform routine tests because it's not the main purpose of this report.

Now let's do the same thing with our random sample in R. I also dropped outliers in "time". For the simple regression I got

```
> lm1=lm(dif~time, data=tab)
> summary(lm1)

Coefficients:
              Estimate Std. Error t value        Pr(>|t|)
(Intercept) 244.951504 2.656289  92.22 <0.0000000000000002 ***
time          1.598878  0.002822 566.67 <0.0000000000000002 ***
```

and for the model with two predictors

```
> lm2=lm(dif~time+sur, data=tab)
> summary(lm2)

Coefficients:
              Estimate Std. Error t value         Pr(>|t|)
(Intercept) 232.761474 3.058566 76.102 < 0.0000000000000002
time          1.600113  0.002825 566.368 < 0.0000000000000002
sur           0.354324  0.044097  8.035  0.00000000000000094
```

Thus, the coefficients are almost exactly the same. The standard errors are smaller for the the whole data than for the random sample, as expected. But the orders of these standard errors are rather close (keep in mind that standard errors are square roots of variances).This means that my both approaches match very well.

The next graph represents fitted values vs observed values according to the model with one predictor ("Model 1") and to the model with two predictors ("Model 2"). Any of the two my approach can be used (with `C++` or with sampling) to get the above plot since both approaches give almost exactly the same coefficients.

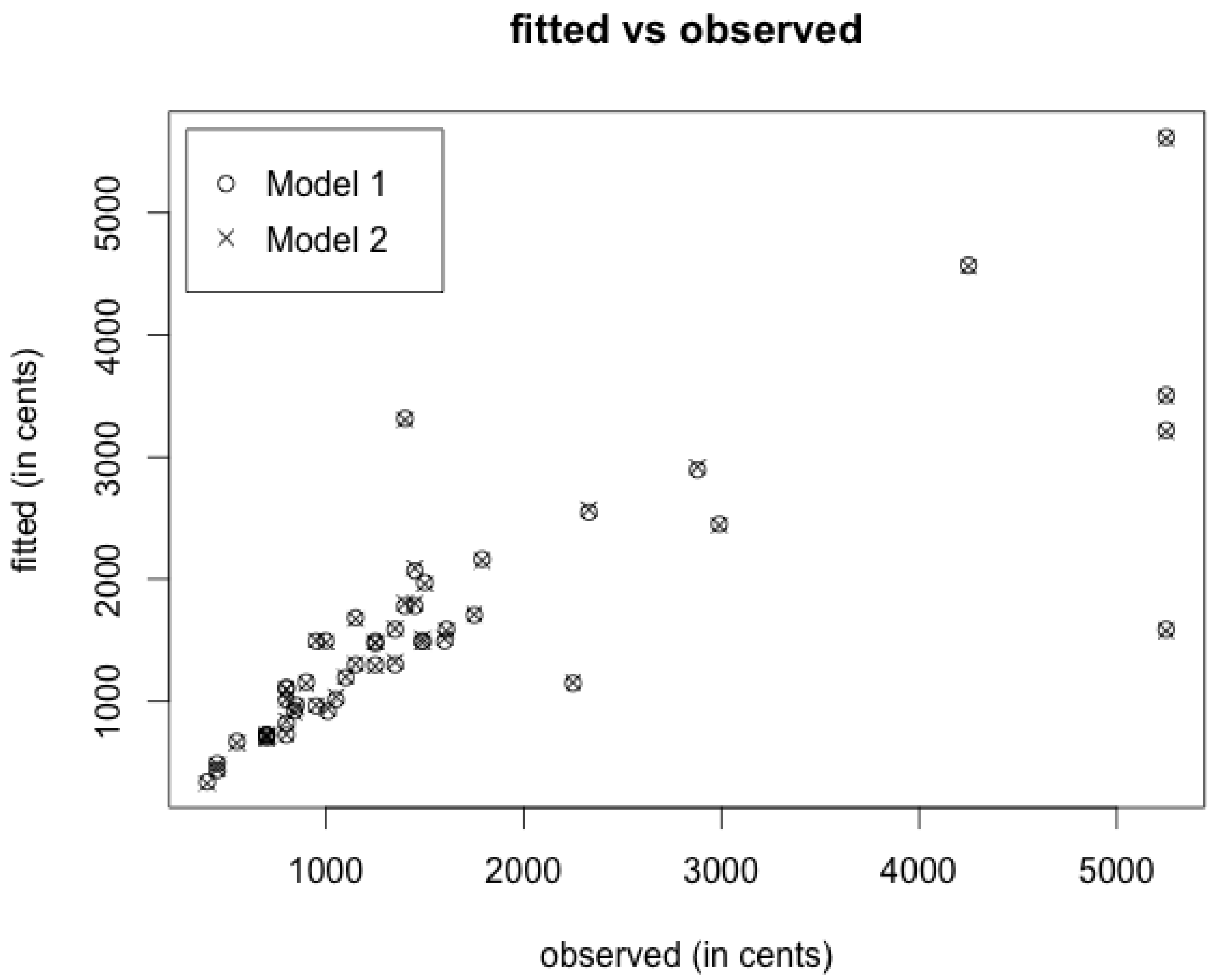

Figure 3

Both models give almost the same fitted values according to this plot.

Compare two approaches. Let me first list some good aspects both approaches share.

- use a single computer
- doesn't require much from the computer (don't need fast GPU, for instance)
- took small amount of time for the computer to perform

The bad aspects:

- took a lot of my time
- not very flexible

Now why using `C++` was better?

- processed the whole data, not a sample from it
- you don't need to think about theory: about randomness, independence and so on

On the other hand sampling was better is the following.

- we can do multiple tasks
- we can do very difficult tasks without thinking how to enhance the performance
- we can do these tasks extremely fast
- quite flexible for some purposes

I strongly believe that if one needs to do a hardcore analysis and a lot of plotting then he has to use sampling. Also, I noticed that it can be more convenient to use `awk` instead of `C++` for simple tasks; `awk` and `C++` work, roughly speaking, equally fast. For more complicated tasks it was easier for me to use `C++`.