Laboration: Robustness and Distribution Assumptions

Simon Sigurdhsson

September 1, 2010

Abstract

Analyzing distribution assumptions qualitatively and quantitatively to decide what distribution real-world data may or may not have; especially concerning (assumed) normally distributed data.

1 Introduction

In statistics, when assuming data is of a certain distribution, one must be able to test and confirm that hypothesis in order to form more elaborate theories concerning the data. To do this, several tools are available; qualitative analysis such as plot comparison of estimated variables, quantitative analysis such as the χ^2 goodness-of-fit-test, and more. Doing this computationally is preferred, since calculations are many and fairly intensive; MATLAB is the tool of choice in this laboration.

2 Test of Distribution Assumptions

Loading the `ibm.txt` file into MATLAB yields a (rather large) structure. We are interested in the seventh column exclusively, an therefore quite simply extract it. If needed, the `ibm` structure can be freed, as we no longer need it. We also reverse the data order, so that the values obtained are sorted by ascending time. Additionally, we get rid of problematic zero values (which would break the logreturns, since we can't divide by zero). This is done using the MATLAB `find` function:

```
values = ibm(end:-1:1, 7);
values = values(find(values));
```

It is now time to calculate the actual logreturns. Instead of accessing and dividing the elements of our array by some cumbersome loop, we simply perform some very simple vector subtraction (thanks to the fact that $\log(S_t/S_{t-1}) = \log(S_t) - \log(S_{t-1})$); we "shift" the vector so that the desired values can be obtained by simple subtraction. Firs, however, we compute the logarithm. This is how it looks:

```
logs = log(values);
logreturn = logs(2:end) - logs(1:end-1);
logret = logreturn(1:1000);
```

Plotting the cumulative sum of the logreturns (all of them) yields the plot seen in figure 1.

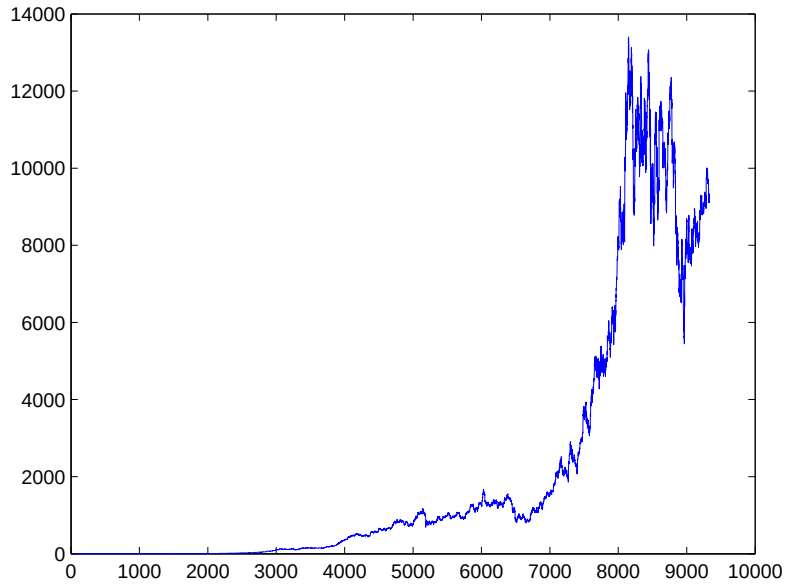

Figure 1: A plot of the cumulative sum of all logreturns.

2.1 Qualitative analysis

Performing ML estimates is very easy in MATLAB. Using the `mle` function with suitable input gives a good estimation of the parameters of the distribution tested against. To estimate μ and σ^2 in the case of a normal distribution, we simply specify that we're assuming a normal distribution:

```
params = mle(logreturn, 'distribution', 'normal');
plot([-0.5:0.01:0.5], normpdf([-0.5:0.01:0.5], params(1), params(2)), 'r');
```

The variable `params` now contains two elements; the first is $\hat{\mu}$ and the second $\hat{\sigma}^2$. In later parts of this laboration, we will use these variables to estimate the actual parameters. The second line in the code fragment above simply plots the estimated normal density using the ML-estimated parameters. The result (along with the kernel density estimate described below) can be seen in figure 2.

Performing a kernel density estimation is also easy; using the `ksdensity` function, that by default plots the estimated density, we get the blue function in figure 2. The code used looks like this (most of it is adjusting the plot).

```
hold on; ksdensity(logreturn); axis([-0.6 0.6 0 50]);
```

Another way of performing a qualitative analysis is plotting the test data in a qq-plot and comparing this to a reference qq-plot. This takes some effort in MATLAB. To begin with, you can plot a 45deg line (as a kind of weak graphic reference) and set up suitable variables; a sequence $\frac{i}{n}$ to go along with the (sorted) data samples X_i, $F^{-1}(\frac{i}{n})$, and reference data Y_i and $F^{-1}(\frac{i}{n})$. This is how:

```
plot([-0.2 0.2], [-0.2 0.2], 'k'); hold on;
```

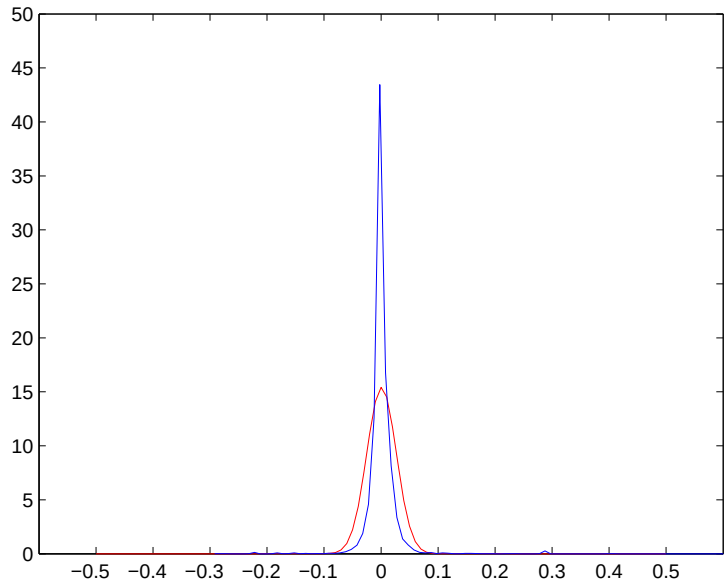

Figure 2: A plot of the estimated density using ML estimates (red) and kernel density estimates (blue).

```
datasample = sort(logreturn);
sequence = (1:length(datasample))/length(datasample);
qqx = datasample;
qqy = norminv(sequence, params(1), params(2));
refx = sort(normrnd(params(1), params(2), [1 100]));
refy = norminv((1:100)/100, params(1), params(2));
```

In this code fragment, `qqx` represents X_i, `qqy` represents the accompanying $F^{-1}(\frac{i}{n})$, `refx` represents Y_i and `refy` is the equivalent of `qqy`, only shorter since Y_i contains less samples than X_i. Plotting these with the code below yields what we can see in figure 3 — which implies that normal distribution is fairly accurate. However, the data plot looks more like a cumulative distribution function, which suggests that the data might not be normally distributed after all (since it deviates from the reference plot). The actual plot code looks like this:

```
plot(qqx, qqy, 'go');
plot(refx, refy, 'rx');
axis([-0.2 0.2 -0.2 0.2]);
```

2.2 Quantitative analysis

A quantitative analysis can easily be performed in MATLAB; a χ^2 goodness-of-fit analysis can be made using the `chi2gof` function. Passing a copy of the `normcdf` function with the parameters we acquired from the ML estimates earlier makes the `chi2gof` function respect these parameters instead of estimating new ones. This is how the function is used in this case:

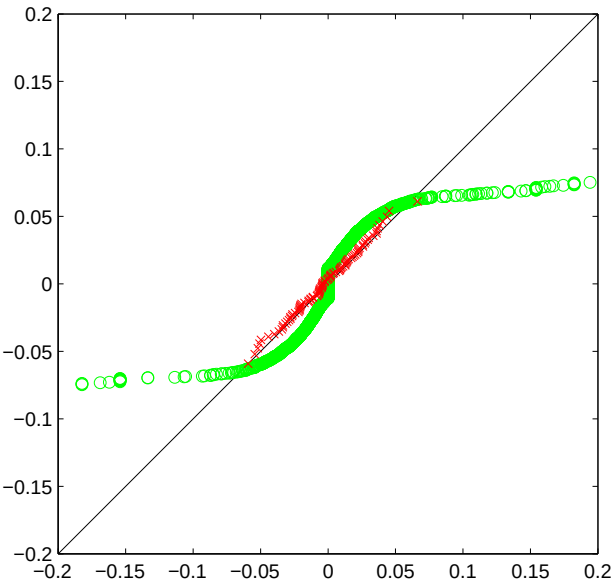

Figure 3: A qq-plot of the data using a normal distribution assumption (green) and a reference plot (red).

```
reject = chi2gof(logreturn, 'cdf', {@normcdf, params(1), params(2)})
```

For the data given in this laboration, `chi2gof` returns 0, meaning the null hypothesis cannot be rejected with a 5% confidence interval. A return value of 1 means the null hypothesis probably should be rejected.

3 Effect of Distribution Assumption

Generating 100 normal distributed values using MATLAB is easy; the `normrnd` function does exactly this. Calculating a confidence interval for these values is also easy — the `normfit` function can (if required) return a confidence interval after fitting the data to an estimated normal distribution with parameters $\hat{\mu}$ and $\hat{\sigma}^2$. This is what it actually looks like, in MATLAB:

```
rvs = normrnd(0, 1, [1 100]);
[muhat, sigmahat, muci, sigmaci] = normfit(rvs);
```

Checking that these are inside the interval is trivial; simply incrementing a counter if they are is also trivial. Additionally, the σ confidence interval edges are stored in an array, on which we later use the `mean` and `diff` functions to calculate an average confidence interval width. What we're actually doing is averaging the edges, then calculating the difference:

```
average_interval_length = diff(mean(sigmacis'));
```

Running this several timed showed that between 93% and 96% of the time, the real σ was indeed inside the confidence interval. The average interval length hovered around 0.283. This is to be expected at a 95% confidence interval, as σ is "correct" roughly that often, and the interval length corresponds to roughly 5% deviance on either side of the estimated $\hat{\sigma}$.

Doing the same thing with a Student-t distribution is not difficult; replacing `normrnd` with `trnd` (and correct parameters, of course) generates random data according to the Student-t distribution. In all other aspects, the code remains essentially the same. The results obtained (with a parameter $\nu = 4$) are as follows: 6% of the time, the original σ was inside the confidence interval. Less than 1% of the time, $\tilde{\sigma} = \nu = 3$ was inside the confidence interval. This clearly indicates that the data isn't normal distributed; if we were dealing with real data, we'd have to reject the null hypothesis of normal distribution. The interval width is larger as well, hovering around 0.4, which further indicates that something is wrong (as it is too large with respect to the 95% confidence interval).

Setting $\nu = 3$ further reduces the number of σs inside the confidence interval, down to 1%. The interval length at this value is close to 0.5. Increasing ν to 10 has an interesting effect; the confidence interval width is reduced to roughly 0.3, and the number of σs inside the confidence interval increases to 63%! Further increasing the parameter has similar results; at $\nu = 100$ the results are very similar to the results obtained with actual normal distributed data. This is quite simply because of the following relation:

$$\lim_{\nu \to \infty} X = Y, \quad X \sim t(\nu), \quad Y \sim N(0,1)$$

4 Robust Estimation

To perform a robust estimation of the contaminated distribution as given, we first observe that in the equation $X = WY + (1-W)Z$, we have three different distributions:

$$Y \sim N(0, \sigma_Y) Z \sim Cauchy(0) = Student(1) W \sim Bernoulli(1-\epsilon) = Binomial(1, 1-\epsilon)$$

Hence, we need to generate random samples from these distributions and combine these to get X. The following code does just that:

```
W = binornd(1, 0.95, [1 100]);
Z = trnd(1, [1 100]);
Y = normrnd(0, 1, [1 100]);
X = W.*Y + (1-W).*Z;
```

Calculating the expected value of this distribution, given the generated data, is now simply a matter of using `mean` on our data vector `X`. Storing each value of the 1000-iteration loop in a vector, we can then sort this vector and finally obtain the 25th and 975th values, which are -0.56 and 0.58 respectively. Additionally, we can generate a histogram, seen in figure 4, which displays the problematic nature of this contaminated distribution — some values deviate heavily from the normally distributed uncontaminated data.

To fix this, one can use robust estimators when estimating the expected value. Doing this takes a little more effort — instead of simply using the built-in `mean` function, we

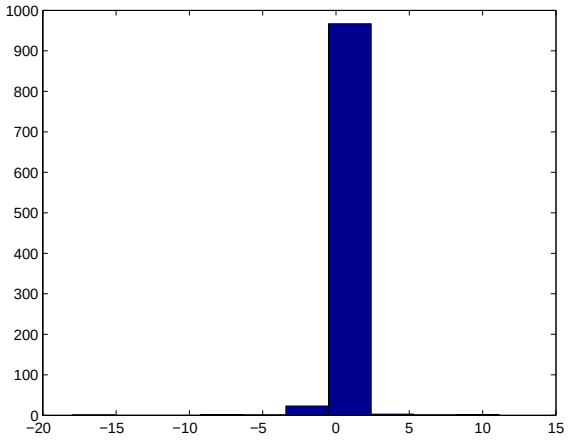

Figure 4: A histogram plot of the expected values of the contaminated data

have to construct a more complicated expression. Taking $k = \alpha n$ and giving this value to the variable `k`, i.e. `k=0.1*100`, we can use this expression:

```
robust = (median(X) + X((k+1):(100-k))/(100-2*k))/2;
```

Using this value in later calculations yield the histogram seen in figure 5, and 25/975 values of -0.36 and -0.16 respectively. Compared to the straight expected value, these values are much closer together, and the histogram is far more concentrated. This implies that the robust estimator is much better at dealing with contaminated distributions, at least when calculating the expected value. Other robust methods are (by definition, really) likely to be better than their non-robust equivalent as well.

This has the side-effect of discarding data, but when some data deviates very much, it is likely to be irrelevant (or contaminated) and should thus be discarded.

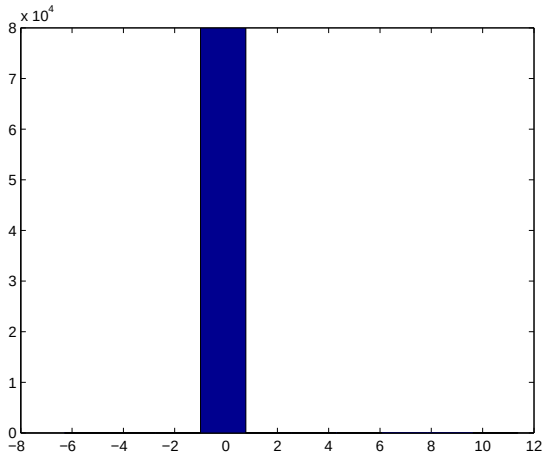

Figure 5: Histogram plot after correction by robust estimators