Regression Analysis
Assignment 5: Robust Regression

Due on December 2, 2016 sharp 3pm

MAT 3375

Professor Termeh Kousha

Gabriel Lalonde

7546220

Eric Rozon

7801836

1 Introduction

When performing multiple linear regression analysis, Ordinary Least Squares (OLS) works fantastically well under a certain set of assumptions. In particular, it is often assumed that:

1. the error terms are normally distributed with constant variance (homoskedasticity), and
2. (relatively) large outliers are not present.

The process used in class to derive the OLS estimates can be generalized to include a wider class of estimates. In his 2002 paper, Fox introduces the concept of the M-estimation.

2 Definitions

Before proceeding, it is important to define certain terms.

Definition 1. If our model (drawn from a population of size n) is $\hat{Y}_i = \hat{\beta} X_i$ then the error term is defined to be $e_i := Y_i - \hat{Y}_i$.

Definition 2. An objective function is a function $\rho : \mathbb{R} \to \mathbb{R}_{\geq 0}$ with the following properties:

- $\rho(0) = 0$
- $\rho(e) \geq 0$
- $\rho(e) = \rho(-e)$
- $\rho(e) > \rho(e')$ whenever $|e| > |e'|$

Example. In the OLS Model, we define $\rho(e) = e^2$. However, this choice is not unique. One could just as well choose the absolute error $\rho(e) = |e|$ to satisfy the criteria. More abstractly, Fox details the Huber and Bisquare objective functions in his paper.

Definition 3. The weight function $w : \mathbb{R} \to \mathbb{R}_{\geq 0}$ defined by

$$w(e) := \frac{\rho'(e)}{e}$$

where ρ' denotes the derivative of ρ. It is a measure of the influence of the error e in the derivation of the regression coefficient matrix $\hat{\beta}$.In the ordinary least squares case, the weight function is constant at 2 for every e; every error is equally weighted (that is to say, since only relative weights matter, we can standardize to $w(e_i) = 1$ for all i).

Remark 1. From our weight function and objective function we derive the estimates by solving the system of equations

$$\sum_{i=1}^{n} w(e_i) \cdot e_i \cdot X_i = 0$$

This method is familiar to us from the OLS parameter derivations; however, Fox details that since there can be cross dependency between the weights, the residuals, and the estimated coefficients, one often must use a process called iteratively reweighted least squares, which is very technical and boring and is therefore skipped in this exposition. The case where $\rho(e) = |e|$ highlights an interesting example.

Example. First, notice that if $\rho(e) = |e|$ then it is differentiable everywhere but at $e = 0$, and its dirivative is

$$\rho'(e) = \frac{|e|}{e}$$

from which we notice that

$$w(e) = \frac{|e|}{e^2} = \frac{1}{|e|}$$

This difference in weighting from OLS leads to what would seem to be an important observation. While the weight of all the errors is constant in OLS, the weight of the error terms decreases as e gets further from 0. That is to say that the absolute error is less reactive to large outliers than is OLS; one deficiency of this objective function is that the weight is not defined when $e = 0$.

Definition 4. Given a set of data with n observations and a model $Y = X\beta + \epsilon$, the breakdown point is defined to be the minimal number of outliers which could possibly make the model invalid/useless. For example, in OLS, the breakdown point is $\frac{1}{n}$ since one significant outlier can bend the entire regression line away from what would otherwise be sensible. Different weighting can change this result so as to make a model more robust to outliers.

Example. Assuming one unique error is larger than all the rest, one could derive the regular OLS estimates but instead use the weight function:

$$w(e_k) = \begin{cases} 1 & |e_k| < \max\limits_{i=1}^{n} |e_i| \\ 0 & |e_k| = \max\limits_{i=1}^{n} |e_i| \end{cases}$$

which yields a breakdown point of $\frac{2}{n}$ since this weight function simply drops the most significant outlier.

Definition 5. Assuming again that $Y = X\beta + \epsilon$, one can always derive the OLS estimator model $Y = X\hat{\beta}$; suppose now that an alternative model is proposed, call it $Y = X\gamma$. We define the efficiency of the new model by

$$\text{Efficiency}_\gamma = \frac{MSE_{\hat{\beta}}}{MSE_\gamma}$$

where $MSE_{\hat{\beta}}$ and MSE_{γ} are the mean squared errors of the model with $\hat{\beta}$ and γ respectively. Usually then (but not always), $0 \leq$ Efficiency ≤ 1 and an effiency of close to 1 is desirable for reasons detailed by Montgomery.

Remark 2. Note that in general, there is a tradeoff between having a high breakdown point and a high level of efficiency. In general, an increase in one entails a decrease in the other, so one must decide what is more important in an individual instance. This intuitively makes sense since one must decide whether unusual seeming data points can be ignored, or whether they may be indicative of a significant trend which must be taken into account. This is shown in more detail in our example.

3 Technical Details

As previously mentioned, Fox details both the Huber and the Bisquare estimators in his paper. In all their glory, their respective objective functions and weight functions follow.

3.1 Huber Estimator

The Huber estimator requires that one fix a tuning constant $k > 0$. The general objective function (with variable k) for the Huber estimator is

$$\rho_{Huber} = \begin{cases} \frac{1}{2}e^2 & |e| \leq k \\ k|e| - \frac{1}{2}k^2 & |e| > k \end{cases}$$

while the weight function for the Huber estimator is

$$w_{Huber} = \begin{cases} 1 & |e| \leq k \\ k/|e| & |e| > k \end{cases}$$

3.2 Bisquare Estimator

Similarly to the Huber estimator, a tuning constant $k > 0$ must be fixed. The general objective function for the Bisquare estimator is

$$\rho_{Bisquare} = \begin{cases} \frac{k^2}{6}\left(1 - \left(1 - (\frac{e}{k})^2\right)\right) & |e| \leq k \\ \frac{k^2}{6} & |e| > k \end{cases}$$

and the weight function for the Bisquare estimator is

$$w_{Bisquare} = \begin{cases} \left(1 - (\frac{e}{k})^2\right)^2 & |e| \leq k \\ 0 & |e| > k \end{cases}$$

4 Crime Dataset

To display the usefulness of the concepts explored above, we use a dataset from the United States which reports on different variables relating to crime as measured on a state by state basis. The relevant variables to this discussion, with their name in the dataset in parentheses, are:

- Violent crimes per 100,000 people (Y);
- percentage of the population in a metropolitan area (X_1);
- percentage of the population that is white (X_2);
- percent of population with a high school education or above (X_3)
- percent of population living under poverty line (X_4); and
- percent of population that are single parents (X_5).

The data is presented on every state as well as Washington DC, so there are $n = 51$ observations in this set. The data was gathered from

http://www.ats.ucla.edu/stat/data/crime.dta

as referenced by the Institute for Digital Research and Education, UCLA.

Let's get started with the importation of the data.

```
> library(foreign)
> data = read.dta("crime.dta")
> data = data[c("crime","pctmetro","pctwhite","pcths","poverty","single")]
> names(data)=c("y","x1","x2","x3","x4","x5")
> y=data$y
> x1=data$x1
> x2=data$x2
> x3=data$x3
> x4=data$x4
> x5=data$x5
```

There are 32 possible variable selections. Since it isn't a large number of possibilities, we will fit all the possible regressions.

```
> library(leaps)
> allsubsets=regsubsets(y~x1+x2+x3+x4+x5,nbest=10,data)
> s1<-summary(allsubsets)
data.frame(s1$outmat,SSE=s1$rss,R2=s1$rsq,adjR2=s1$adjr2,Cp=s1$cp)
```

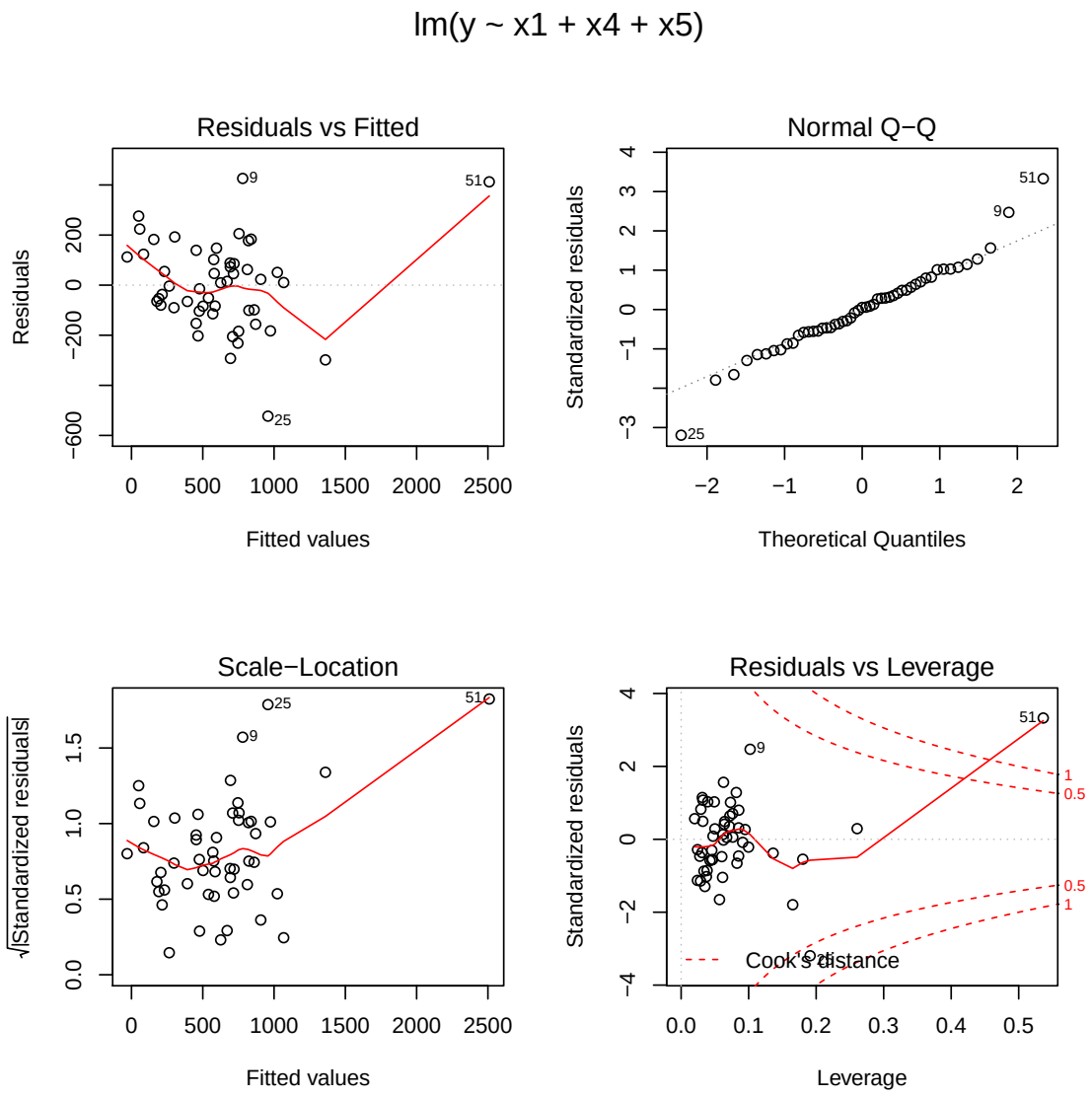
		x1	x2	x3	x4	x5	SSE	R2	adjR2	Cp
1	(1)					*	2882441	0.70371089	0.69766418	41.804747
1	(2)		*				5267318	0.45856690	0.44751725	115.280112
1	(3)	*					6849058	0.29597824	0.28161045	164.011720
1	(4)				*		7202979	0.25959837	0.24448814	174.915615
1	(5)			*			9090650	0.06556264	0.04649249	233.072643
2	(1)	*				*	1773079	0.81774336	0.81014934	9.626559
2	(2)		*			*	2608856	0.73183297	0.72065935	35.375904
2	(3)			*		*	2829916	0.70910997	0.69698955	42.186519
2	(4)				*	*	2848602	0.70718922	0.69498877	42.762212
2	(5)	*			*		3975976	0.59130530	0.57427635	77.495322
2	(6)	*	*				4173437	0.57100805	0.55313339	83.578879
2	(7)		*		*		4574065	0.52982716	0.51023662	95.921754
2	(8)		*	*			5259429	0.45937787	0.43685194	117.037047
2	(9)	*		*			6221958	0.36043846	0.33379006	146.691491
2	(10)			*	*		6873436	0.29347239	0.26403374	166.762783
3	(1)	*			*	*	1557995	0.83985213	0.82962992	5.000048
3	(2)	*	*			*	1689119	0.82637366	0.81529113	9.039858
3	(3)	*		*		*	1689536	0.82633088	0.81524562	9.052681
3	(4)		*		*	*	2583280	0.73446201	0.71751278	36.587921
3	(5)		*	*		*	2599934	0.73275008	0.71569157	37.101027
3	(6)			*	*	*	2829730	0.70912910	0.69056287	44.180785
3	(7)	*	*		*		2983166	0.69335724	0.67378430	48.907979
3	(8)	*		*	*		3474748	0.64282702	0.62002875	64.053061
3	(9)		*	*	*		4076762	0.58094543	0.55419726	82.600417
3	(10)	*	*	*			4124758	0.57601181	0.54894873	84.079138
4	(1)	*	*		*	*	1500336	0.84577892	0.83236839	5.223650
4	(2)	*		*	*	*	1545329	0.84115404	0.82734135	6.609834
4	(3)	*	*	*		*	1642714	0.83114368	0.81646052	9.610172
4	(4)	*	*	*	*		2399197	0.75338404	0.73193917	32.916548
4	(5)		*	*	*	*	2580919	0.73470469	0.71163553	38.515186
5	(1)	*	*	*	*	*	1460618	0.84986152	0.83317947	6.000000

There are three models with Cp between p and 2p that looks promising. There is $Y = X_1 + X_4 + X_5$,$Y = X_1 + X_2 + X_4 + X_5$ and $Y = X_1 + X_2 + X_3 + X_4 + X_5$ where their Cp statistic is 5.0000, 5.2237 and 6.0000 respectively. The third model that has Cp=p is obviously the full model, but the goal of this analysis is to check what is the smallest model that still does the job well without adding too much bias. It is a good idea to reduce the model here to not overfit our data or get multicolinearity. We have choosen the model $Y = X_1 + X_4 + X_5$ since it is the smallest one that still has a minimal amount of bias.
Just to be on the safe side, we will do a multicolinearity check.

```
> library(car)
> lm=lm(y~x1+x4+x5)
> vif(lm)
      x1       x4       x5
1.144807 1.527117 1.631659
```

Multicolinearity is not a problem. Now let's analyse this model.

```
> pdf("ols.pdf",height=7,width=7)
> par(mfrow = c(2,2), oma = c(0, 0, 1.1, 0))
> plot(lm)
> dev.off()
```



Given the cook's distance graph, sample 25 and 51 are outliers. The standardized residuals graph doesn't show any problematic patterns. The qq-plot is heavy-tailed meaning that our data set contains extremes that are outside of the Normal distribution. In other words, there

are outliers. Samples 9 is probably problematic.

4.1 Finding the robust regressions

Firstly, we will find the Huber Robust regression model.

```
> library(MASS)
> rlm.huber=rlm(y~x1+x4+x5,data=data,psi=psi.huber)
> summary(rlm.huber)

Call: rlm(formula = y ~ x1 + x4 + x5, data = data, psi = psi.huber)
Residuals::
     Min       1Q   Median       3Q      Max
-556.886  -91.966    4.484   97.611  478.691

Coefficients:
            Value      Std. Error t value
(Intercept) -1572.1291   135.2622   -11.6228
x1              6.8948     1.1479     6.0066
x4             19.4760     6.3499     3.0671
x5            127.2305    14.1831     8.9706
> #let's get T(0.025,49) to check the t value of the regressors
> abs(qt(0.025,47))
[1] 2.011741
> #get the 15 smallest weights.
> data.frame(resid=rlm.huber$resid, weight=rlm.huber$w
+ )[order(rlm.huber$w),][1:15,]
        resid    weight
25 -556.88611 0.3377288
51  478.69090 0.3929858
9   441.59864 0.4259286
18 -292.88002 0.6421215
39 -263.44268 0.7139697
12  250.46056 0.7509303
14  224.94344 0.8361726
20  214.61056 0.8764274
47 -209.39385 0.8982353
48 -201.99999 0.9311438
1    48.29970 1.0000000
2    85.38794 1.0000000
3   106.04620 1.0000000
4  -136.27802 1.0000000
```

```
5    38.55966 1.0000000
```

All regressor are valid with a confidence level of 95%. Samples 9,25 and 51 are the samples we tought were problematic in the OLS analysis. Those three samples are the most down-weighted ones by the Huber weighting function. Then there are 7 other samples that are slightly reduced. The rest of the samples kept all there weight. This looks like a very good model that handled properly the outliers. Let's see now how things differs with bisquare weighting.

```
> rlm.bisquare=rlm(y~x1+x4+x5,data=data,psi=psi.bisquare)
Warning message:
In rlm.default(x, y, weights, method = method, wt.method = wt.method,   :
  'rlm' failed to converge in 20 steps
> #Since the rlm didn't converge, let's give it more steps.
> rlm.bisquare=rlm(y~x1+x4+x5,data=data,psi=psi.bisquare,maxit=40)
> summary(rlm.bisqure)

Call: rlm(formula = y ~ x1 + x4 + x5, data = data, psi = psi.bisquare,
    maxit = 40)
Residuals:
    Min      1Q  Median      3Q     Max
-638.51 -107.15  -25.86   99.48  444.19

Coefficients:
            Value      Std. Error t value
(Intercept) -1704.2518   140.4320   -12.1358
x1              6.5479     1.1917     5.4944
x4             21.3231     6.5926     3.2344
x5            139.3921    14.7252     9.4662

Residual standard error: 155.8 on 47 degrees of freedom
> #All the weights in increasing order
> data.frame(resid=rlm.bisquare$resid, weight=rlm.bisquare$w
+ )[order(rlm.bisquare$w),]
         resid     weight
25 -638.513036 0.05515601
9   444.190424 0.39646535
18 -364.712227 0.56308487
51  327.967349 0.63680630
12  269.297360 0.74630910
39 -250.884153 0.77766500
14  221.225660 0.82470968
```

```
20  215.068316 0.83389721
47 -213.120171 0.83676666
48 -196.008408 0.86097093
35 -190.361918 0.86859479
13  186.257471 0.87401470
22 -185.731789 0.87471461
16  183.422599 0.87769537
29  182.014928 0.87951363
6  -162.108773 0.90378498
40  156.944092 0.90965304
4  -150.374740 0.91692521
49 -145.108709 0.92252130
24  138.355040 0.92942725
19  131.813769 0.93584267
44 -124.287874 0.94284348
10 -115.999529 0.95013692
21 -109.221377 0.95571738
31  105.876761 0.95837055
46 -105.085335 0.95897484
28  104.147238 0.95970820
43  -98.965031 0.96357904
26  -98.046522 0.96424201
27   94.821605 0.96652963
3    86.603520 0.97203637
33   86.562334 0.97206606
41   85.716816 0.97261134
37  -77.843877 0.97738234
45  -73.782415 0.97966398
2    68.892811 0.98225702
17  -62.868566 0.98522205
34   57.831739 0.98747791
7   -55.487872 0.98847032
38  -52.638493 0.98962222
23  -49.846747 0.99069148
42   47.835262 0.99142464
8    42.175798 0.99333383
15  -41.153683 0.99365189
30  -40.198876 0.99393896
11   37.071471 0.99485240
32  -27.063069 0.99725752
36  -25.858571 0.99749359
```

```
5    18.589051 0.99870109
50    6.747469 0.99982895
1     4.202759 0.99993311
```

All regressors in the resulting model are valid with a confidence level of 95%. With the bisquare weighting function, the weigth are far more "severe". All samples are weighted down. Sample 25 is almost gone and samples 9 and 18 have been moderatly lightened. However, sample 51 is a severe outlier and we don't think that the bisquare weigthing function has sufficiently attenuated its influence on the model.

5 Tuning of the Weighting Functions

Disclaimer: Altough this section's name might make an impression of "these guys knows what they are doing", please note that this is just an attempt to optimize the weighting functions for our dataset using our limited knowledge on the subject. The following approach will be intuition based and not be rigourous. Now let's have some fun.
In order to tune those functions, we will run simulations to see how our tunings influence the efficiency and the breakdown point of the weighting functions. We will try to make our simulations data as close as possible to the real datas, since we want to specifically tune those functions for the crime dataset.

```
> lm$coefficients
 (Intercept)           x1           x4           x5
-1666.435892     7.828935    17.680244   132.408052
> (summary(lm)$sigma)**2
[1] 33148.82
```

The simulation dataset that we will be successively recreated will be of size $n = 51$ and be generated by this distribution obtained from the OLS made in the crime data example: $Y = -1666.435892 + 7.828935X_1 + 17.680244X_4 + 132.408052X_5 + N(0, 33148.82)$. We won't regenerate X_1, X_4 and X_5 since it will make the simulation further from the real dataset without much improvement in the randomness of the simulation samples.

5.1 Tuning of the Huber Weighting Function

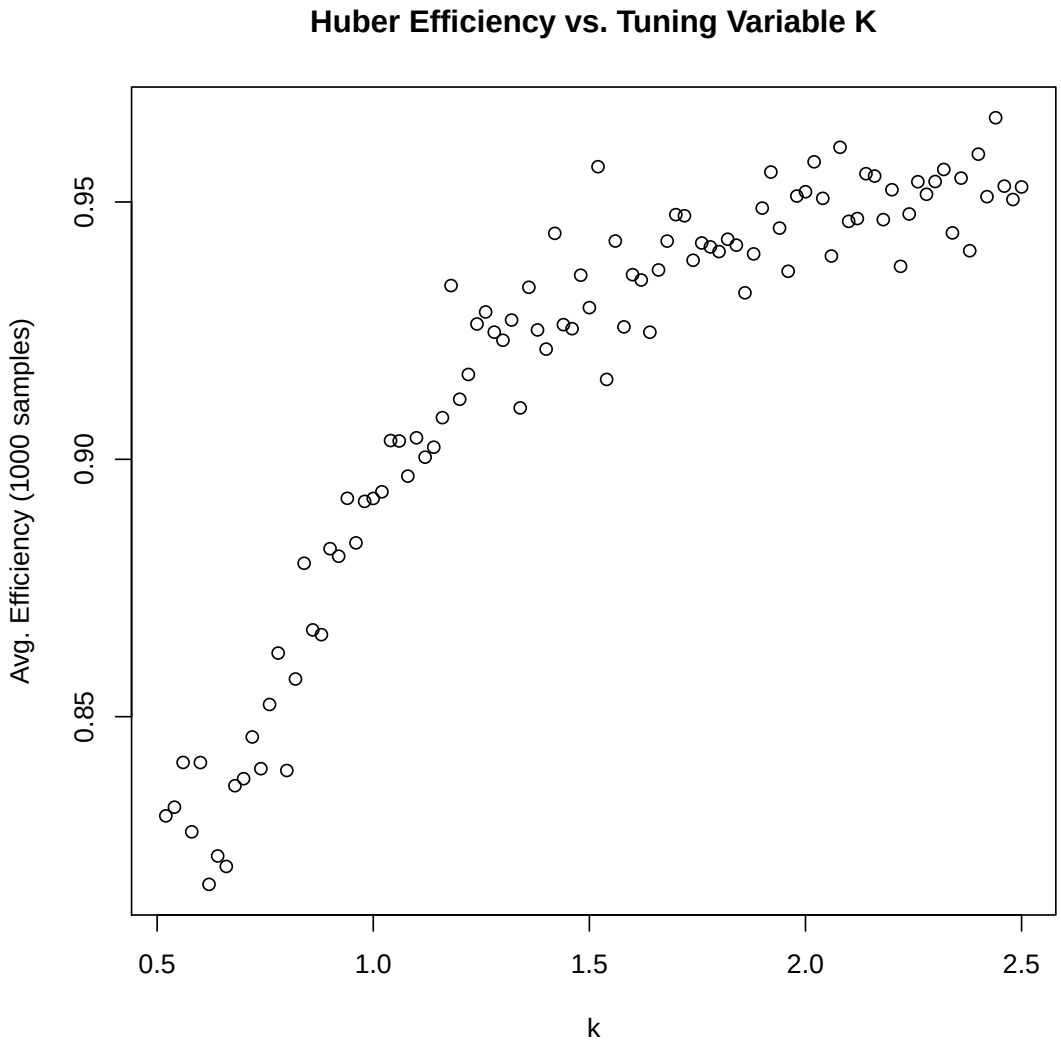
Let's first check how the efficiency reacts to changes in k. What we will measure here is the finite sample efficiency that is defined by $\frac{MSE_{RLM}}{MSE_{OLS}}$. We made the following R script to take 1000 samples of 100 different k values from 0.5 to 2.5. (Below 0.5, many iterations does't completes.)

```
> avg.eff=matrix(1,100)
> k=matrix(1,100)
>
```

```
> #scan thru all k's
> for(j in 1:100)
+ {
+     #generates k based on j
+     kj=0.5+j/50
+     efficiencies=matrix(1,1000)
+     #make new dataset
+     noresid=-1666.435892+7.828935*data$x1+
+     17.680244*data$x4+132.408052*data$x5
+     #Do 1000 samples per k
+     for(i in 1:1000){
+         #create OLS
+         new.y=noresid+rnorm(51,0,182.0682)
+         new.lm=lm(new.y~x1+x4+x5)
+         lm.mse=anova(new.lm)$"Mean Sq"[4]
+         #create RML
+         new.rlm=rlm(new.y~x1+x4+x5,
+          data=data.frame(new.y=new.y,x1=data$x1,x4=data$x4,x5=data$x5),
+          psi=psi.huber,k=kj,maxit=100)
+         rlm.mse=(summary(new.rlm)$sigma)**2
+         #compute the efficiency
+         efficiencies[i]=rlm.mse/lm.mse
+     }
+     k[j]=kj
+     #make the mean of efficiencies for a given k
+     avg.eff[j]=mean(efficiencies)
+ }
```

This has given us the following results.

```
> pdf("efficiencyHuber.pdf",height=7,width=7)
> plot(avg.eff~k,ylab="Avg. Efficiency (1000 samples)",
+ main="Huber Efficiency vs. Tuning Variable K")
> dev.off()
```



In this plot, we can see that there is a lot of variance in the efficiency even though each point is the average of 1000 samples. This might be due to the small size of the dataset. We can see that the efficiency increase with k. It looks likes that after $k = 2$. the efficiency becomes steady. There is still the breakdown point response to check in order to make our choice of k.

Statisticians sure do have delicate formulas to obtain a precise breakdown point value, but as of Dec 2nd, 4:43 am, we (well "I") will cut corners a bit. This time, we will generate a dataset, find the RLM regression, add a random obvious outlier in the first or fourth quartile, redo the RLM regression, check if the new model is still "close" to the original one and repeat until we obtain the breakdown in j steps. This process is repeated 1000 times for every k value, we do the average of the j's. The Y in the crime dataset varies between 326.5 and 2922.0, therefore a $Y_i = 6000$ in the first or fourth quartile should be considered as an outlier. This will be the outlier value we will use. The breakdown criteria will be the the t-test failure with 95% confidence of one regressor or the divergence of the RLM regression.

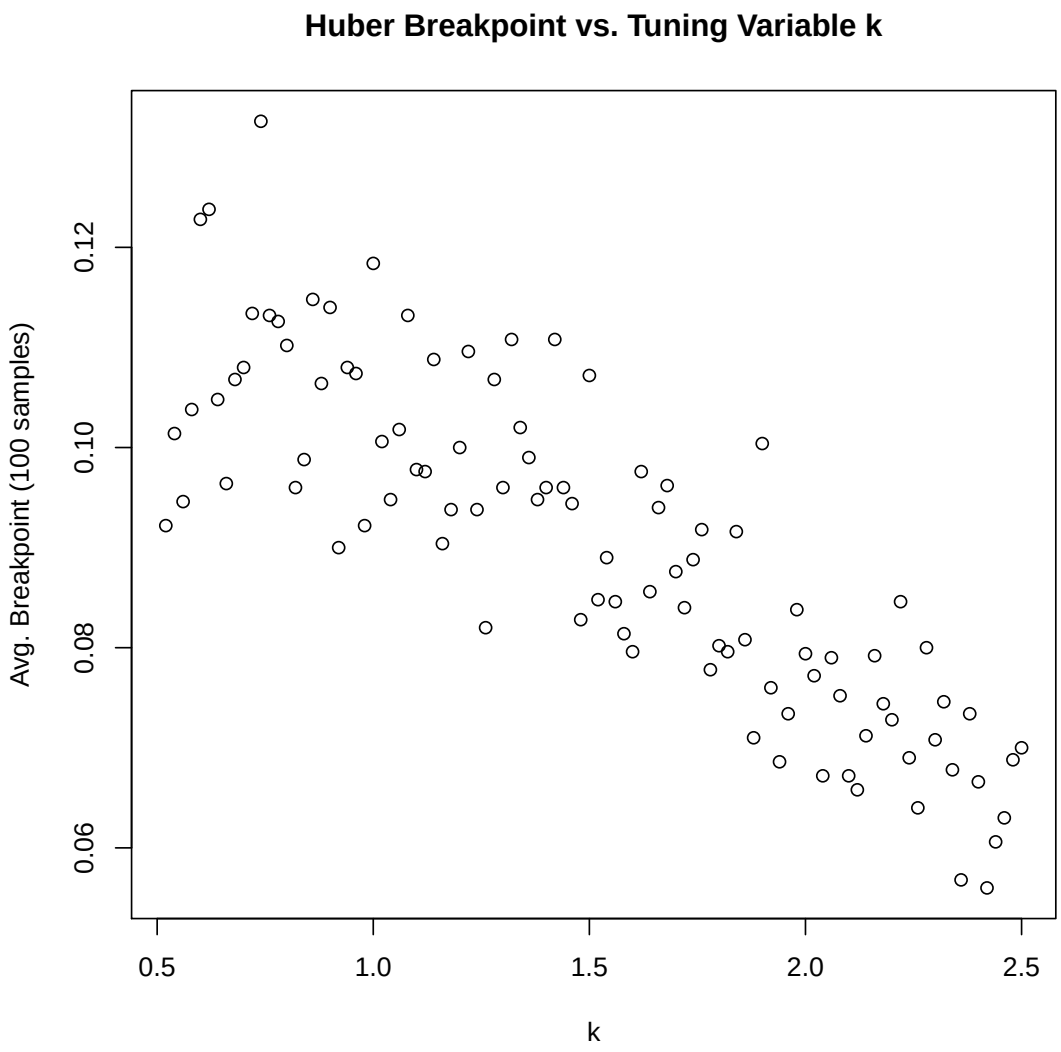
This is the script.

```
> avg.break=matrix(1,100)
> k=matrix(1,100)
> #samples every k
> for(j in 1:100){
+ breakpoint=matrix(1,100)
+ #do 100 samples for every k
+ for(i in 1:100){
+     #generates k given j
+     a=0.5+j/50
+     t.value=qt(0.05/2,47,lower=F)
+     #generate new datas
+     new.data=data.frame(new.y=-1666.435892+7.828935*data$x1+
+      17.680244*data$x4+132.408052*data$x5+rnorm(51,0,182.0682),
+      new.x1=data$x1, new.x4=data$x4, new.x5=data$x5)
+     #order the data to be able to do quartile insertions
+     new.data=new.data[order(new.data$new.y),]
+     #generate a random sequence of indexes
+     #in the 1st and 4th quartile
+     outSequence=c(1,2,3,4,5,6,7,8,9,10,11,12,13,
+      40,41,42,43,44,45,46,47,48,49,50,51)
+     outSequence=outSequence[sample.int(length(outSequence))]
+     #make the "perfect" RML for later comparison
+     rlm.orig=rlm(new.y~new.x1+new.x4+new.x5,new.data,
+      psi=psi.huber,k=a,maxit=100)
+     count=0
+     #This loop until the model breakdown
+     repeat{
+         #insert the outlier
+         new.data$new.y[outSequence[count]]=6000
+         #update the RML model
+         rlm.dyn=rlm(new.y~new.x1+new.x4+new.x5,new.data,
+          psi=psi.huber,k=a,maxit=100)
+         #check if the model is in breakdown state
+         if(count>=25 || !rlm.dyn$converged||
+             min(abs(summary(rlm.dyn)[[4]][,3]))<t.value){
+             break
+         }
+         count=count+1
+     }
```

```
+     breakpoint[i]=count
+ }
+ k[j]=a
+ #de average for avery k
+ avg.break[j]=mean(breakpoint)
+ }
  #make the breakdown a percentage
> avg.break=avg.break/51
```

It produces the following results.

```
> pdf("breakpointHuber.pdf",height=7,width=7)
> plot(avg.break~k,ylab="Avg. Breakpoint (100 samples)",
+ main="Huber Breakpoint vs. Tuning Variable k")
> dev.off()
```



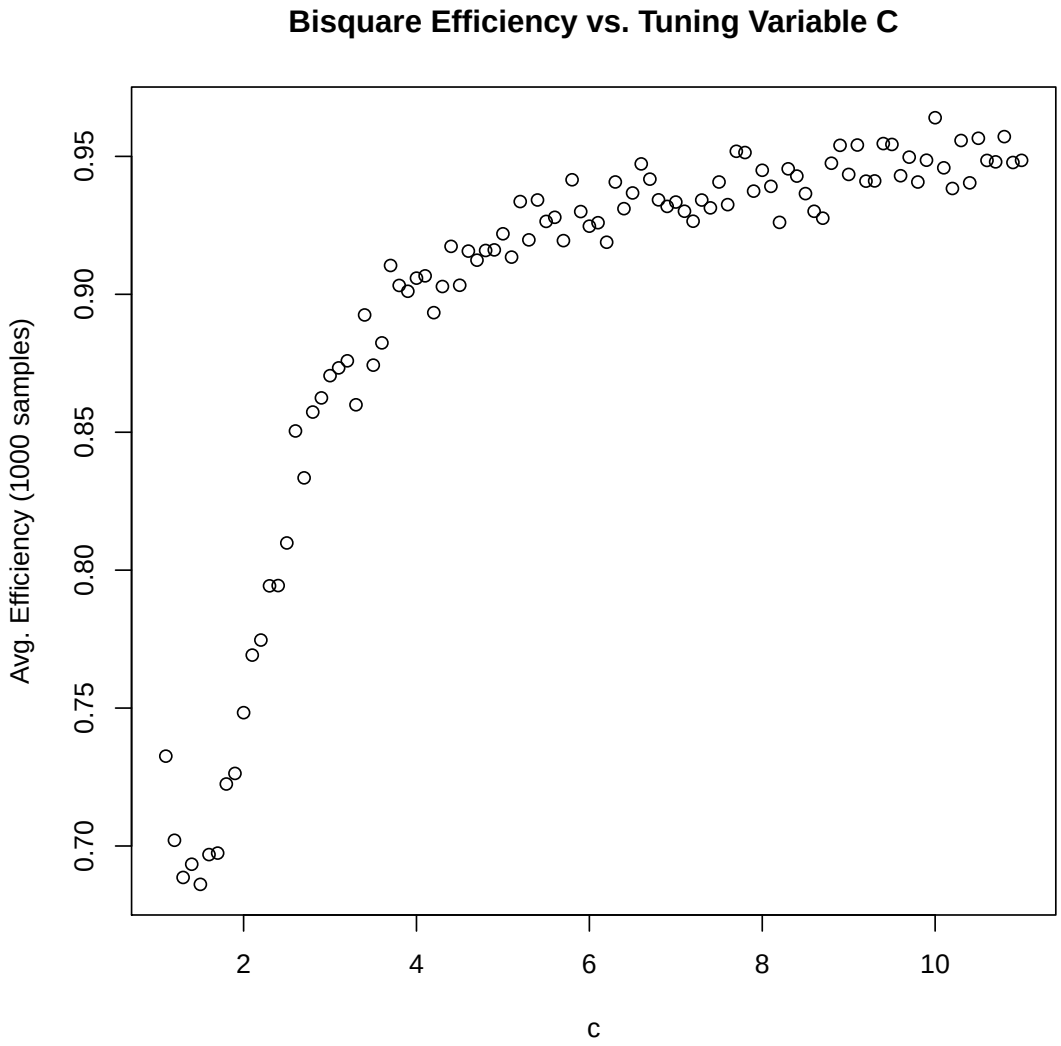
Once again the variance is important. As the value k increase, the breakpoint decrease. In the current situation, we have 2 outliers in the crime dataset and maybe 1 or two potential outliers. Therefore, a breakpoint of 10% (can handle up to 5 outliers) is ideal in our case. This corresponds roughly to $k = 1.5$. This k value give us an efficiency arround 92.5% according to our previous table which is decent. Let's see how the bisquare RLM compares.

5.2 Tuning of Tuney's Bisquare Weighting Function

We will apply the same techniques to the bisquare weigthing function since it behave similarly and has also only one tuning variable. Here's the execution of the script.

```
> avg.eff=matrix(1,100)
> c=matrix(1,100)
>
> #scan thru all c's
> for(j in 1:100)
+ {
+     #generates c based on j
+     cj=1 +j/10
+     efficiencies=matrix(1,1000)
+     #make new dataset
+     noresid=-1666.435892+7.828935*data$x1+
+     17.680244*data$x4+132.408052*data$x5
+     #Do 1000 samples per c
+     for(i in 1:1000){
+         #create OLS
+         new.y=noresid+rnorm(51,0,182.0682)
+         new.lm=lm(new.y~x1+x4+x5)
+         lm.mse=anova(new.lm)$"Mean Sq"[4]
+         #create RML
+         new.rlm=rlm(new.y~x1+x4+x5,
+          data=data.frame(new.y=new.y,x1=data$x1,x4=data$x4,x5=data$x5),
+          psi=psi.bisquare,c=cj,maxit=100)
+         if(!new.rlm$converged){
+             print(cj)}
+         rlm.mse=(summary(new.rlm)$sigma)**2
+         #compute the efficiency
+         efficiencies[i]=rlm.mse/lm.mse
+     }
+     c[j]=cj
+     #make the mean of efficiencies for a given k
+     avg.eff[j]=mean(efficiencies)
```

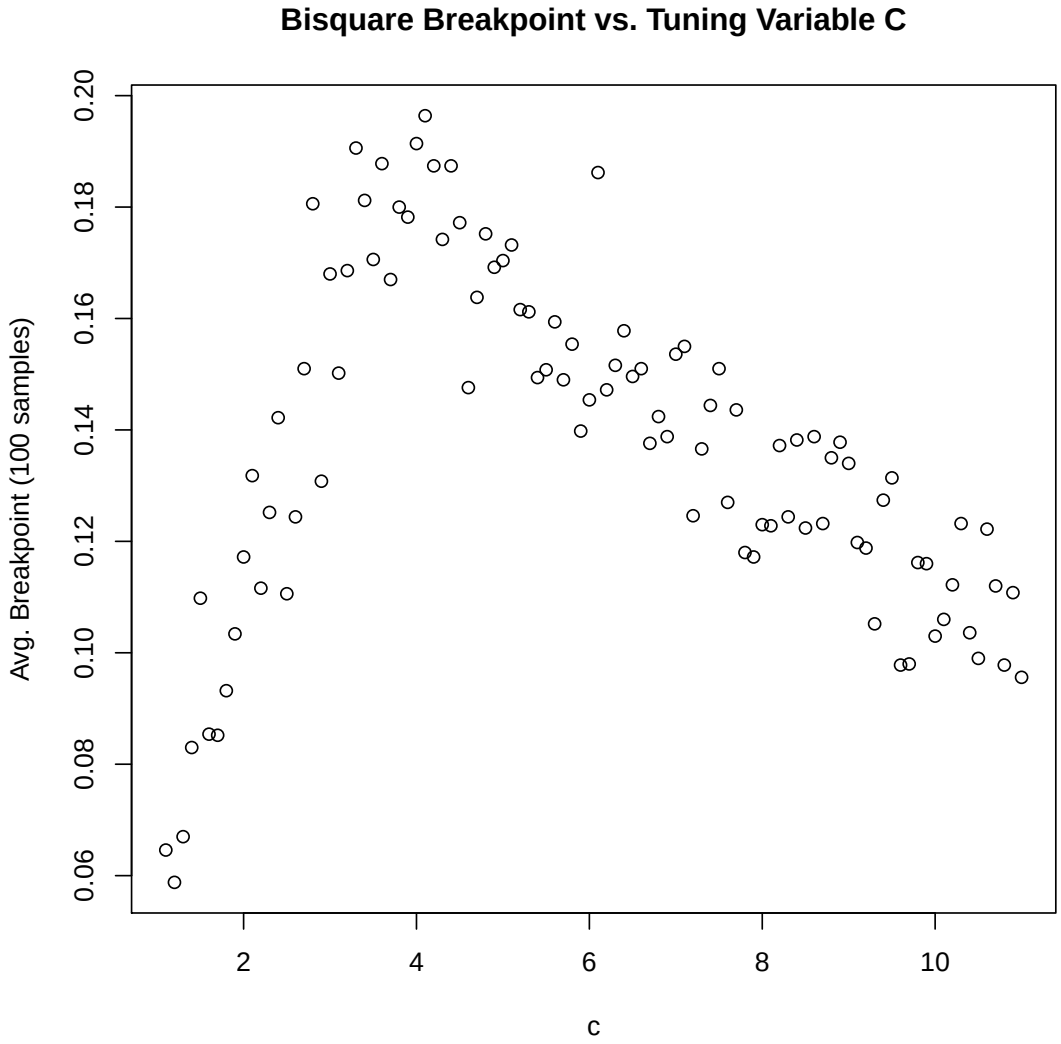
```
+ }
[1] 1.3
[1] 1.4
[1] 1.4
[1] 1.8
[1] 2.1
[1] 2.2
[1] 2.3
[1] 5.4
[1] 6.4
> pdf("efficiencyBisquare.pdf",height=7,width=7)
> plot(avg.eff~c,ylab="Avg. Efficiency (1000 samples)",
+ main="Bisquare Efficiency vs. Tuning Variable C")
> dev.off()
```



The efficiency increase with the variable c. From $c = 1.5$ to $c = 6$ the growth is agressive but for $c > 6$, the slope is almost horizontal. Let's check the breakpoint response as we did previously.

```
> avg.break=matrix(1,100)
> c=matrix(1,100)
> #samples every c
> for(j in 1:100){
+ breakpoint=matrix(1,100)
+ #do 100 samples for every c
+ for(i in 1:100){
+     #generates c given j
+     a=1+j/10
+     t.value=qt(0.05/2,47,lower=F)
+     #generate new datas
```

```
+     new.data=data.frame(new.y=-1666.435892+7.828935*data$x1+
+      17.680244*data$x4+132.408052*data$x5+rnorm(51,0,182.0682),
+      new.x1=data$x1, new.x4=data$x4, new.x5=data$x5)
+     #order the data to be able to do quartile insertions
+     new.data=new.data[order(new.data$new.y),]
+     #generate a random sequence of indexes
+     #in the 1st and 4th quartile
+     outSequence=c(1,2,3,4,5,6,7,8,9,10,11,12,13,
+      40,41,42,43,44,45,46,47,48,49,50,51)
+     outSequence=outSequence[sample.int(length(outSequence))]
+     #make the "perfect" RML for later comparison
+     rlm.orig=rlm(new.y~new.x1+new.x4+new.x5,new.data,
+      psi=psi.bisquare,c=a,maxit=100)
+     count=0
+     #This loop until the model breakdown
+     repeat{
+         #insert the outlier
+         new.data$new.y[outSequence[count]]=6000
+         #update the RML model
+         rlm.dyn=rlm(new.y~new.x1+new.x4+new.x5,new.data,
+          psi=psi.bisquare,c=a,maxit=100)
+         #check if the model is in breakdown state
+         if(count>=25 || !rlm.dyn$converged||
+             min(abs(summary(rlm.dyn)[[4]][,3]))<t.value){
+             break
+         }
+         count=count+1
+     }
+     breakpoint[i]=count
+ }
+ c[j]=a
+ #do average for avery c
+ avg.break[j]=mean(breakpoint)
+ }
> #generate the pdf plot
> pdf("breakpointBisquare.pdf",height=7,width=7)
> #make the breakdown a percentage
> avg.break=avg.break/50
> plot(avg.break~c,ylab="Avg. Breakpoint (100 samples)",
+ main="Bisquare Breakpoint vs. Tuning Variable C")
> dev.off()
```



The breakpoint response to c for the bisquare weighting function forms a "mountain" where the summit is arround $c = 4$. It is evident that the bisquare function is more robust than the Huber one. Now, let's select the perfect c for our dataset. We only need a break point of at least 10%. The bigger c is, the most efficient the weighting will be. Therefore we have choosen $c = 10$. This gives us a breakpoint around 10% and an efficiency of roughly 95% which is really good.

6 Tuning Improvements

A comparison between the default RML model and the tuned one is made here in order to see if the many hours of analysis we did were useful.

6.1 Huber MLR comparison

```
> rlm.huber2=rlm(y~x1+x4+x5,data,psi=psi.huber,k=1.5,maxit=100)
> #previously obtained weights
> data.frame(resid=rlm.huber$resid,weight=rlm.huber$w
+ )[order(rlm.huber$w),][1:15,]
        resid    weight
25 -556.88611 0.3377288
51  478.69090 0.3929858
9   441.59864 0.4259286
18 -292.88002 0.6421215
39 -263.44268 0.7139697
12  250.46056 0.7509303
14  224.94344 0.8361726
20  214.61056 0.8764274
47 -209.39385 0.8982353
48 -201.99999 0.9311438
1    48.29970 1.0000000
2    85.38794 1.0000000
3   106.04620 1.0000000
4  -136.27802 1.0000000
5    38.55966 1.0000000
> #Tuned weights
> data.frame(resid=rlm.huber2$resid,weight=rlm.huber2$w
+ )[order(rlm.huber2$w),][1:15,]
        resid    weight
25 -553.96427 0.3767017
51  473.07348 0.4411765
9   441.74348 0.4724198
18 -291.82818 0.7150476
39 -265.89386 0.7848578
12  251.65131 0.8292456
14  223.35880 0.9343253
47 -211.62314 0.9861178
20  210.76155 0.9901668
1    45.22131 1.0000000
2    86.07243 1.0000000
3   109.47739 1.0000000
4  -137.68640 1.0000000
5    37.24425 1.0000000
6  -160.43328 1.0000000
```

We can see that marginal points attenuation is less severe. In our opinion, the tuned Huber

model is better than the untuned one since it is only weighting down (mostly) the three points we had already identified as problematic.

6.2 Tukey's Bisquare MLR Comparison

```
> #previously obtained weight
> data.frame(resid=rlm.bisquare$resid,weight=rlm.bisquare$w
+ )[order(rlm.bisquare$w),][1:15,]
       resid     weight
25 -638.5130 0.05515601
9   444.1904 0.39646535
18 -364.7122 0.56308487
51  327.9673 0.63680630
12  269.2974 0.74630910
39 -250.8842 0.77766500
14  221.2257 0.82470968
20  215.0683 0.83389721
47 -213.1202 0.83676666
48 -196.0084 0.86097093
35 -190.3619 0.86859479
13  186.2575 0.87401470
22 -185.7318 0.87471461
16  183.4226 0.87769537
29  182.0149 0.87951363
> #Tuned weight
> rlm.bisquare2=rlm(y~x1+x4+x5,data,psi=psi.bisquare,c=10,
+ maxit=100)
> data.frame(resid=rlm.bisquare2$resid,weight=rlm.bisquare2$w
+ )[order(rlm.bisquare2$w),][1:15,]
       resid    weight
25 -539.0396 0.7584004
9   430.6479 0.8419280
51  413.8742 0.8534785
18 -303.3980 0.9198587
39 -284.2390 0.9294671
12  271.0763 0.9357479
47 -226.6980 0.9548373
13  214.2085 0.9596274
14  209.9291 0.9612052
48 -201.9518 0.9640715
35 -201.4901 0.9642359
16  189.2411 0.9684188
```

```
20  186.1606 0.9694281
40  181.4795 0.9709327
22 -179.7002 0.9715031
```

Now it feels that the bisquare approach is too conservative. The two confirmed outliers (25 and 51) are barely attenuated. We don't think that this version of the bisquare in this case is an improvement. The un-tuned version wasn't a success either.

7 Conclusion

The previous example, in all of its glory, highlights the importance of understanding when and how robust regression should be used. While it is certainly possible to derive a least squares model which ignores outliers by removing them from one's dataset, it is much more efficient and practical to simply use a robust regression sush as the Huber or Bisquare to generate a more realistic model than would be obtained using Ordinary Least Squares. A word of caution, however; regression analysis is a careful balance of art and science. One must be prudent to keep in mind the practical implications of eliminating/downweighting "unwanted" datapoints. While this certainly makes ones model work more nicely, it is often unwise to remove points simply because they don't fit with the trend(s) in the rest of the data. Nonetheless, Robust Regression is an important tool when used effectively.

8 References

Douglas C. Montgomery, Elizabeth A. Peck, & G. Geoffrey Vining. Introduction to Linear Regression Analysis, 5th Edition ISBN: 978-0-470-54281-1

Fox, John. Robust Regression. Appendix to *An R and S-PLUS Companion to Applied Regression* January 2002.
http://www.saedsayad.com/docs/RobustRegression.pdf

R Data Analysis Examples: Robust Regression.
http://www.ats.ucla.edu/stat/r/dae/rreg.htm