Least Squares Absolute Value Regression

Jan de Leeuw

First created on March 15, 2020. Last update on October 29, 2021

Abstract

We introduce a nonlinear least squares regression problem, tentatively called **least squares absolute value regression**, and develop a convergent MM algorithm to minimize the residual sum of squares. We also discuss the combinatorial aspects of the optimization problem, and introduce a smoothed version.

Contents

Note: This is a working paper which will be expanded/updated frequently. All suggestions for improvement are welcome. The directory deleeuwpdx.net/pubfolders/lsav has a pdf version, the complete Rmd file with all code chunks, the bib file, and the R source code.

1 Introduction

In *least absolute value regression* the regression coefficients are chosen to minimize the sum of the absolute values of the residuals. See, for example, Dielman (2005) for a fairly recent overview.

In this note we consider a different regression problem involving absolute values. It minimizes

$$f(\beta) = (z - |X\beta|)'U(z - |X\beta|) \tag{1}$$

over $\beta \in \mathbb{R}^p$. The absolute values in $|X\beta|$ are defined element-wise. The matrices $X \in \mathbb{R}^{n\times p}$, the positive semi-definite $U \in \mathbb{R}^{n\times n}$, and the vector $z \in \mathbb{R}^n$ are known and fixed.

This is a nonlinear weighted least squares problem, and to confuse the hell out of everybody we call it *least squares absolute value regression* (or *LSAV regression*).

Note that we did not assume that $z \geq 0$, although this would make sense because we are approximating using the non-negative $|X\beta$. As we shall show below, both theory and algorithms simplify if U is diagonal, but again we do not assume this (and we do not assume X is of full column-rank).

2 One-sided Directional Derivatives

Necessary conditions for a local minimum of the LSAV loss function can be derived from a formula for the (one-sided) directional derivatives. First

$$g_i(\beta,\delta) = \frac{|x_i'(\beta+\epsilon\delta)| - |x_i'\beta|}{\epsilon} = \begin{cases} |x_i'\delta| & \text{if } x_i'\beta = 0, \\ \text{sign}(x_i'\beta)x_i'\delta & \text{if } x_i'\beta \neq 0. \end{cases}$$

And thus

$$\mathcal{D}f(\beta,\delta) = \lim_{\epsilon\downarrow 0}\frac{f(\beta+\epsilon\delta) - f(\beta)}{\epsilon} = -2(z - |X\beta|)'Ug(\beta,\delta).$$

It follows that a necessary condition for f to have a local minimum at β is $(z - |X\beta|)'Ug(\beta,\delta) \leq 0$ for all δ.

Using this necessary condition we can now derive a result similar to one in De Leeuw (1984) for multidimensional scaling.

Theorem 1: Necessary Conditions:

If

1. $z > 0$,
2. U is diagonal and positive definite,

and f has a local minimum at β, then $x_i'\beta \neq 0$ for all i.

Proof: The necessary condition for a local minimum can be written as

$$\sum_{i \in \mathcal{I}_0} u_i z_i |x_i'\delta| + \sum_{i \in \mathcal{I} - \mathcal{I}_0} u_i(z_i - |x_i'\beta|)\text{sign}(x_i'\beta)x_i'\delta \leq 0.$$

If the second term is negative it can be made positive by changing the sign of δ. Under conditions 1 and 2 the first term is positive if at least one $x_i'\delta \neq 0$. Thus both terms must be zero for all δ, i.e.

1. $\mathcal{I}_0 = \emptyset$,
2. $\sum_{i=1}^n u_i(z_i - |x_i'\beta|)\text{sign}(x_i'\beta)x_i = 0$.

■

3 Combinatorial Considerations

Unidimensional Scaling (UDS, see, for example, De Leeuw (2005)) is a special case of LSAV regression. In UDS the matrix X takes differences of the scale values in β, and thus $|X\beta|$ has the distances between points on the scale. In this section we discuss the properties of UDS shared by LSAV regression.

Divide the index set $\mathcal{I} = \{1, 2, \cdots, n\}$ into three parts, giving a partition $\mathcal{P} = \{\mathcal{I}_0, \mathcal{I}_+, \mathcal{I}_-\}$. Define $\mathcal{K}(\mathcal{P})$ as the (possibly empty) polyhedral convex cone in $\mathbb{R}^p$ consisting of all β such that

$$\begin{aligned} x_i'\beta &= 0 \text{ if } i \in \mathcal{I}_0, \\ x_i'\beta &> 0 \text{ if } i \in \mathcal{I}_+, \\ x_i'\beta &< 0 \text{ if } i \in \mathcal{I}_-. \end{aligned}$$

Also define the matrix $X(\mathcal{P})$ as

$$\begin{aligned} x_i(\mathcal{P}) &= x_i \text{ if } i \in \mathcal{I}_0 \cup \mathcal{I}_+, \\ x_i(\mathcal{P}) &= -x_i \text{ if } i \in \mathcal{I}_-. \end{aligned}$$

Finally

$$f(\beta, \mathcal{P}) = (z - X(\mathcal{P})\beta)'U(z - X(\mathcal{P})\beta),$$

and

$$
\begin{aligned}
f^\star(\mathcal{P}) &= \min_{\beta\in\mathcal{K}(\mathcal{P})} f(\beta,\mathcal{P}),\\
\beta^\star(\mathcal{P}) &= \{\beta\in\mathcal{K}(\mathcal{P})\mid f(\beta,\mathcal{P})=\hat f(\mathcal{P})\}.
\end{aligned}
$$

For the LSAV regression problem

$$
\min_\beta\ (z-|X\beta|)'U(z-|X\beta|)=\min_{\mathcal{P}} f^\star(\mathcal{P}),
$$

and the LSAV regression solution is any $\beta^\star(\mathcal{P})$ corresponding with the minimizing partition. This analysis also provides us with a sufficient condition for a local minimum.

Theorem 2: Sufficient Conditions: If $\beta^\star(\mathcal{P})$ is in the interior of the cone $\mathcal{K}(\mathcal{P})$ then it is a local minimizer of the LSAV loss function (1).

Proof: On each cone the loss function (1) is a convex quadratic. In the interior of the cone it is differentiable, and if the derivatives vanish we are at a local minimizer. ■

This suggest a simplistic algorithm.

- choose a partition $\mathcal{P}$;
- see if the cone $\mathcal{K}(\mathcal{P})$ is empty;
- if it is, go back to the first step;
- if it is not, minimize $f(\beta,\mathcal{P})$ over $\beta\in\mathcal{K}(\mathcal{P})$;
- if $f^\star(\mathcal{P})$ is the smallest so far, keep $\beta^\star(\mathcal{P})$;
- otherwise, go back to the first step.

I have not tried to program this algorithm, for rather obvious reasons. Although I have efficient code to generate all partitions (De Leeuw (2020)), this is still only practical for relatively small n. The simplistic algorithm can probably can be improved a great deal, but I have no detailed analysis. There is some hope, because we expect that very few of the cones will be non-empty. Thus, unlike in unidimensional scaling, we do not expect thousands of local minima.

Perhaps more promising is this alternative algorithm.

- choose $\tilde\beta\in\mathbb{R}^p$;
- compute the corresponding partition $\mathcal{P}(\tilde\beta)$;
- minimize $f(\beta,\mathcal{P}(\tilde\beta))$ over $\beta\in\mathcal{K}(\mathcal{P}(\tilde\beta))$;
- if $f^\star(\mathcal{P}(\tilde\beta))$ is the smallest so far, keep the minimizer $\hat\beta^\star(\mathcal{P})$;
- otherwise, go back to the first step.

Since the function of β we are minimizing here is constant on each cone, we have to make rather large steps from one β to the next, for example by choosing β at random. If $p=2$ or $p=3$ we can develop strategies to go around the circle or the sphere, for example by using polar coordinates. There will be some additional analysis in a future version of these notes (he says).

Our previous result on necessary conditions allows us to be more precise in the case $z > 0$ and U diagonal.

Theorem 3: Necessary and Sufficient Conditions: If $z > 0$ and U is diagonal then $\beta^\star(\mathcal{P})$ is a local minimizer of the LSAV loss function (1) if and only if it is in the interior of the cone $\mathcal{K}(\mathcal{P})$.

Proof: Combine theorems 1 and 2. ■

In the interior of the cone we have $x_i\beta \neq 0$ for all i, and thus if $z > 0$ and U is diagonal the loss function is differentiable at all local minima.

Our combinatorial algorithms can ignore all partitions with $\mathcal{I}_0 \neq \emptyset$, which leaves "only" 2^n partitions. More importantly perhaps, we can also ignore the linear inequality constraints and the quadratic programming that are part of the algorithms. Just compute the unconstrained minimizer of $f(\beta, \mathcal{P})$, which it is a local minimizer if and only if it is in the cone $\mathcal{K}(\mathcal{P})$.

4 An MM Algorithm

We now develop a majorization or MM algorithm to locally minimize loss function (1). This will only produce local minima, and gives no information about other local minima.

For the general theory of MM algorithms, see De Leeuw (1994), (**deleeuw_B_16b?**) and especially Lange (2016 (in press)).

4.1 Inequalities for Absolute Values

To construct an MM algorithm we need simple bounds at $\tilde{x}$ for the absolute value function. They are

$$\text{sign}(\tilde{x})x \leq |x| \leq \frac{1}{2}\frac{1}{\tilde{x}}(x^2 + \tilde{x}^2).$$

Observe that the lower bound only depends on the sign of $\tilde{x}$, and the upper bound does not work if $\tilde{x} = 0$. We address this problem in a later section.

4.2 Majorizing f

Since $f(\beta) = z'Uz - 2z'U|X\beta| + |X\beta|'U|X\beta|$ we can majorize f by minorizing $z'U|X\beta|$ and majorizing $|X\beta|'U|X\beta|$.

To minorize $z'U|X\beta|$ we define $v = Uz$ and decompose v into its non-negative and negative parts, so that $v = v^+ - v^-$. Now

$$z'U|X\beta| = \sum_{i=1}^n v_i^+|x_i'\beta| - \sum_{i=1}^n v_i^-|x_i'\beta| \geq \sum_{i=1}^n v_i^+\text{sign}(x_i'\tilde{\beta})x_i'\beta - \frac{1}{2}\sum_{i=1}^n \frac{v_i^-}{|x_i'\tilde{\beta}|}\left\{(x_i'\beta)^2 + (x_i'\tilde{\beta})^2\right\}.$$

To majorize $|X\beta|'U|X\beta|$ we start with

$$\begin{aligned}|X\beta|'U|X\beta| &= (|X\tilde{\beta}| + (|X\beta| - |X\tilde{\beta}|))'U(|X\tilde{\beta}| + (|X\beta| - |X\tilde{\beta}|))\\ &= |X\tilde{\beta}|'U|X\tilde{\beta}| + 2|X\tilde{\beta}|'U(|X\beta| - |X\tilde{\beta}|)) + (|X\beta| - |X\tilde{\beta}|))'U(|X\beta| - |X\tilde{\beta}|))\\ &\leq |X\tilde{\beta}|'U|X\tilde{\beta}| + 2|X\tilde{\beta}|'U(|X\beta| - |X\tilde{\beta}|)) + \gamma(|X\beta| - |X\tilde{\beta}|))'(|X\beta| - |X\tilde{\beta}|))\\ &= \gamma\beta'X'X\beta + 2|X\tilde{\beta}|'(U-\gamma I)|X\beta| + \gamma\tilde{\beta}'X'X\tilde{\beta} - |X\tilde{\beta}|'U|X\tilde{\beta}|,\end{aligned}$$

where γ is the largest eigenvalue of U.

Define $w = (U - \gamma I)|X\tilde{\beta}|$, with decomposition $w = w^+ - w^-$. Then

$$|X\tilde{\beta}|'(U-\gamma I)|X\beta| = \sum_{i=1}^n w_i^+|x_i'\beta| - \sum_{i=1}^n w_i^-|x_i'\beta| \leq \frac{1}{2}\sum_{i=1}^n \frac{w_i^+}{|x_i'\tilde{\beta}|}\left\{(x_i'\beta)^2 + (x_i'\tilde{\beta})^2\right\} - \sum_{i=1}^n w_i^-\text{sign}(x_i'\tilde{\beta})x_i'\beta.$$

Collecting terms gives a quadratic majorization of f at $\tilde{\beta}$. It is of the form

$$g(\beta,\tilde{\beta}) = \gamma\beta'X'X\beta - 2\sum_{i=1}^n(v_i^+ + w_i^-)\text{sign}(x_i'\tilde{\beta})x_i'\beta + \sum_{i=1}^n\frac{(v_i^- + w_i^+)}{|x_i'\tilde{\beta}|}(x_i'\beta)^2$$

plus terms independent of β, which we can ignore. If D is a diagonal matrix with diagonal elements

$$d_{ii}(\tilde{\beta}) = \frac{(v_i^- + w_i^+)}{|x_i'\tilde{\beta}|}$$

and e is a vector with elements

$$e_i(\tilde{\beta}) = (v_i^+ + w_i^-)\text{sign}(x_i'\tilde{\beta})$$

then

$$g(\beta,\tilde{\beta}) = \beta'X'(\gamma I + D(\tilde{\beta}))X\beta - 2\beta'X'e(\tilde{\beta}),$$

which is minimized by

$$\hat{\beta} = [X'(\gamma I + D(\tilde{\beta}))X]^{-1}X'e(\tilde{\beta}),$$

using the Moore-Penrose inverse if necessary. This is the majorization algorithm.

Note that the algorithm simplifies dramatically if $z > 0$ and U is diagonal. First we have $v^- = 0$ and thus

$$z'U|X\beta| \geq \sum_{i=1}^n u_iz_i\text{sign}(x_i'\tilde{\beta})x_i'\beta.$$

And second we have $|X\beta|'U|X\beta| = \beta'X'UX\beta$, and no majorization of this term is necessary. Thus

$$g(\beta,\tilde{\beta}) = -2\sum_{i=1}^n u_iz_i\text{sign}(x_i'\tilde{\beta})x_i'\beta + \beta'X'UX\beta$$

plus terms independent of β, and thus

$$\hat{\beta} = (X'UX)^{-1}X'Ue(\tilde{\beta}),$$

where

$$e_i(\tilde{\beta}) = z_i\ \text{sign}(x_i'\tilde{\beta}).$$

Since the update only depends on the signs of $x_i\beta$, this implies that in this special the MM algorithm converges in a finite number of steps (as is the case in unidimensional scaling).

4.3 Smoothed Absolute Values

We can prevent problems with majorizing the absolute value function at zero by using $|x| \sim \sqrt{x^2+\epsilon}$, with a small positive epsilon. See De Leeuw (2018) for some information about this approximation and some alternatives.

The two-sided inequalities become

$$\sqrt{\tilde{x}^2+\epsilon}+\frac{\tilde{x}}{\sqrt{\tilde{x}^2+\epsilon}}(x-\tilde{x})\leq\sqrt{x^2+\epsilon}\leq\frac{1}{2}\frac{1}{\sqrt{\tilde{x}^2+\epsilon}}(x^2+\tilde{x}^2+2\epsilon).$$

The derivations remain exactly the same, although the details of the algorithm change slightly. The only changes are that

$$d_{ii}(\tilde{\beta})=\frac{(v_i^-+w_i^+)}{\sqrt{(x_i'\tilde{\beta})^2+\epsilon}},$$

and

$$e_i(\tilde{\beta})=(v_i^++w_i^-)\frac{x_i'\tilde{\beta}}{\sqrt{(x_i\beta)^2+\epsilon}}.$$

Our R function lsav() in the appendix has ϵ as a parameter. It defaults to zero, which gives the non-smoothed algorithm.

5 Example

Here is an artifical example with $n = 100$ and $p = 3$. Note that we use a z that is non-negative.

```
set.seed(12345)
x <- matrix(rnorm(300), 100, 3)
z <- rnorm(100) ^ 2
```

In the first run we use $U = I$.

```
lsav(x, z, u = diag(100), lbd = 1)
```

```
## itel     1 fold  379.0650 fnew  251.0201
## itel     2 fold  251.0201 fnew  246.2885
## itel     3 fold  246.2885 fnew  230.8050
## itel     4 fold  230.8050 fnew  217.7119
## itel     5 fold  217.7119 fnew  214.3428
## itel     6 fold  214.3428 fnew  212.2809
## itel     7 fold  212.2809 fnew  206.3457
## itel     8 fold  206.3457 fnew  206.3131
## itel     9 fold  206.3131 fnew  206.3131
```

```
## $coef
##                [,1]
## [1,] -0.1622327034
## [2,]  0.6129614600
## [3,] -0.7084470791
##
## $fit
## [1] 206.3130879
```

Now make U singular by setting $U = I - \frac{1}{n}ee'$.

```r
lsav(x, z, u = diag(100) - 1/100, lbd = 1)
```

```
## itel     1 fold  363.1667 fnew  276.0900
## itel     2 fold  276.0900 fnew  237.9265
## itel     3 fold  237.9265 fnew  220.1935
## itel     4 fold  220.1935 fnew  212.7270
## itel     5 fold  212.7270 fnew  208.4292
## itel     6 fold  208.4292 fnew  205.8906
## itel     7 fold  205.8906 fnew  203.7194
## itel     8 fold  203.7194 fnew  203.0186
## itel     9 fold  203.0186 fnew  202.0038
## itel    10 fold  202.0038 fnew  199.9819
## itel    11 fold  199.9819 fnew  198.5057
## itel    12 fold  198.5057 fnew  197.4625
## itel    13 fold  197.4625 fnew  196.7180
## itel    14 fold  196.7180 fnew  196.0610
## itel    15 fold  196.0610 fnew  195.1720
## itel    16 fold  195.1720 fnew  194.6928
## itel    17 fold  194.6928 fnew  194.1218
## itel    18 fold  194.1218 fnew  193.4723
## itel    19 fold  193.4723 fnew  192.9082
## itel    20 fold  192.9082 fnew  192.5548
## itel    21 fold  192.5548 fnew  192.3549
## itel    22 fold  192.3549 fnew  192.2145
## itel    23 fold  192.2145 fnew  192.1169
## itel    24 fold  192.1169 fnew  192.0577
## itel    25 fold  192.0577 fnew  192.0254
## itel    26 fold  192.0254 fnew  192.0160
## itel    27 fold  192.0160 fnew  192.0106
## itel    28 fold  192.0106 fnew  192.0075
## itel    29 fold  192.0075 fnew  192.0055
## itel    30 fold  192.0055 fnew  192.0041
## itel    31 fold  192.0041 fnew  192.0029
```

```
## itel    32 fold  192.0029 fnew  192.0017
## itel    33 fold  192.0017 fnew  192.0005
## itel    34 fold  192.0005 fnew  191.9993
## itel    35 fold  191.9993 fnew  191.9982
## itel    36 fold  191.9982 fnew  191.9974
## itel    37 fold  191.9974 fnew  191.9968
## itel    38 fold  191.9968 fnew  191.9963
## itel    39 fold  191.9963 fnew  191.9959
## itel    40 fold  191.9959 fnew  191.9957
## itel    41 fold  191.9957 fnew  191.9955
## itel    42 fold  191.9955 fnew  191.9954
## itel    43 fold  191.9954 fnew  191.9953
```

```
## $coef
##                 [,1]
## [1,] -0.04948153991
## [2,]  0.29629558863
## [3,] -0.38235452484
##
## $fit
## [1] 191.9953506
```

And make U *really* singular, by using $U = \frac{1}{n}ee'$.

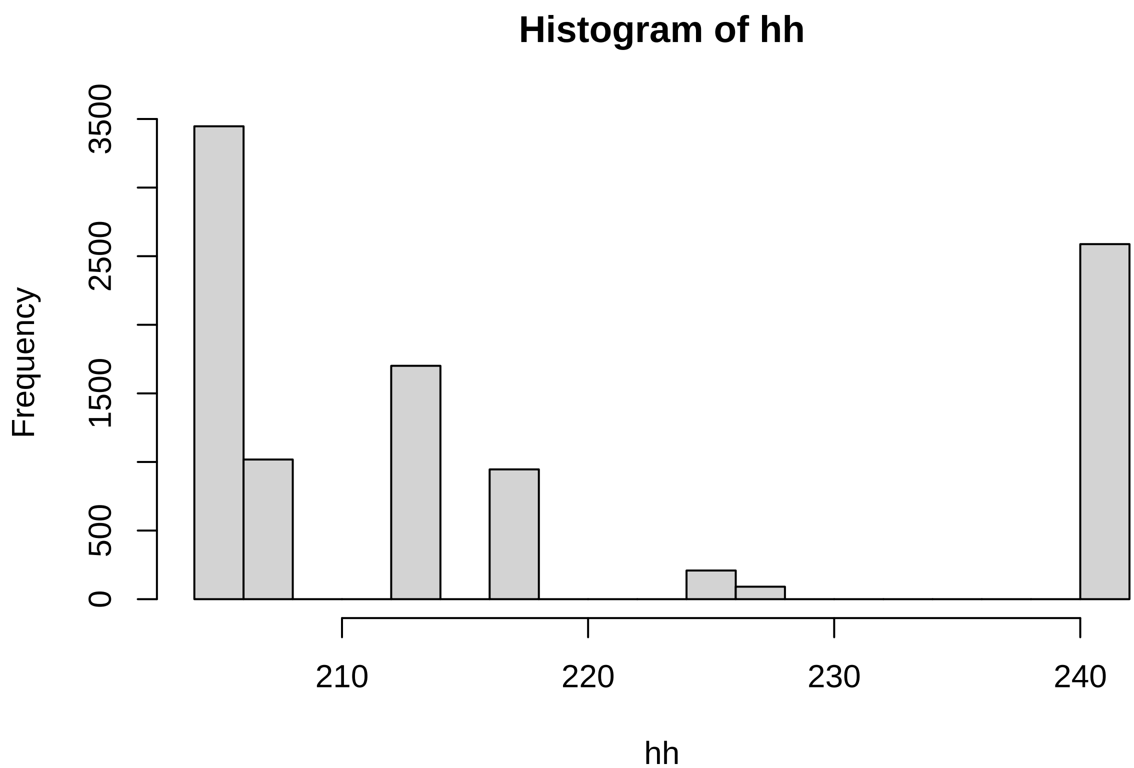
```
lsav(x, z, u = matrix (1/100, 100, 100), lbd = 1)
```

```
## itel     1 fold   15.8983 fnew    2.7457
## itel     2 fold    2.7457 fnew    0.4762
## itel     3 fold    0.4762 fnew    0.0828
## itel     4 fold    0.0828 fnew    0.0144
## itel     5 fold    0.0144 fnew    0.0025
## itel     6 fold    0.0025 fnew    0.0004
## itel     7 fold    0.0004 fnew    0.0001
## itel     8 fold    0.0001 fnew    0.0000
```

```
## $coef
##               [,1]
## [1,] 0.7054162027
## [2,] 0.7150844044
## [3,] 0.7194001311
##
## $fit
## [1] 7.586411332e-05
```

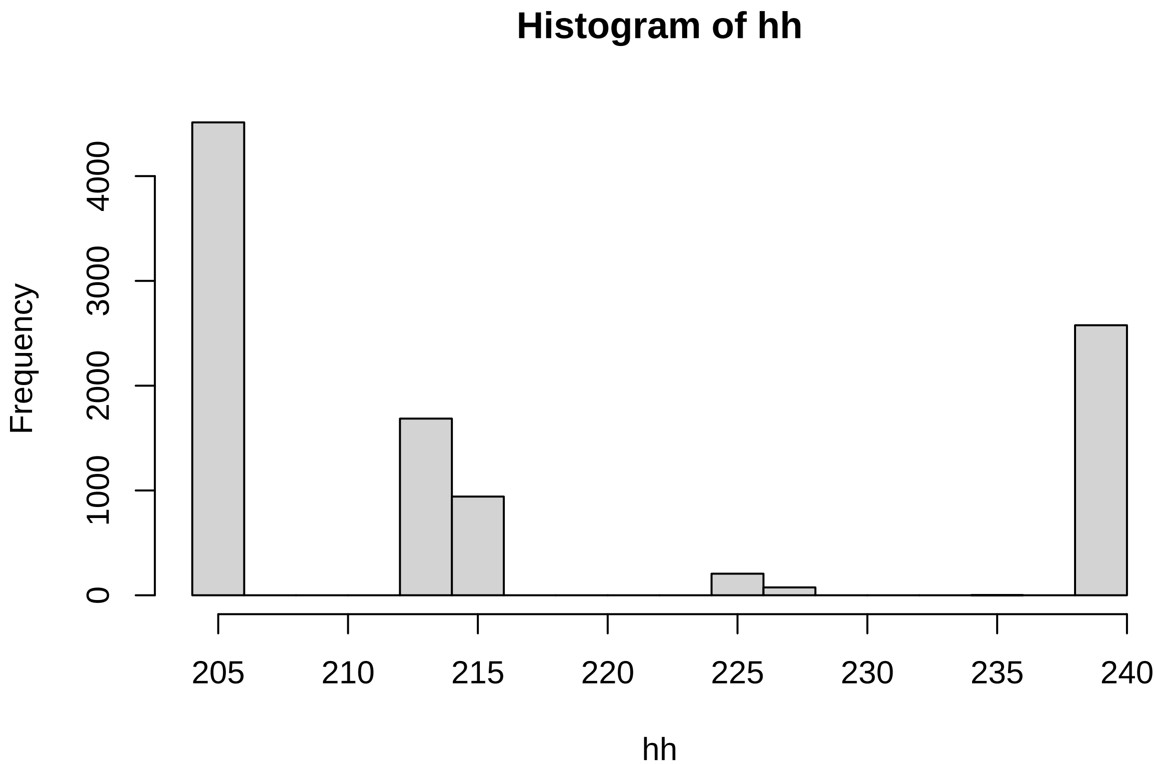
We also looked at local minima, using $U = I$, and 10,000 random starts for β. We find seven of them, but of course the problem will become more serious with a larger number of columns in X (and perhaps less serious if we increase the number of rows for a fixed number of columns). The results may also be quite different if there is some real structure in the data, and not just randomness.

```
set.seed(12345)
hh <- c()
for (i in 1:10000) {
  h <- lsav(x, z, aold = rnorm (3) ^ 2, u = diag(100), lbd = 1, verbose = FALSE)
  hh <- c(hh, h$fit)
}
hist(hh)
```



A small perturbation of U makes the local minima problem less serious.

```
set.seed(12345)
hh <- c()
for (i in 1:10000) {
  h <- lsav(x, z, aold = rnorm (3) ^ 2, u = diag(100)-.001, lbd = 1, verbose = FALSE)
  hh <- c(hh, h$fit)
}
hist(hh)
```



We repeat the computations with smoothing parameter ϵ equal to .01. We cannot expect finite convergence anymore for diagonal U, but that may actually be a good thing. Otherwise the results are, of course, very similar. Here are the results for $U = I$, $U = I - \frac{1}{n}ee'$, and $U = \frac{1}{n}ee'$.

```
set.seed(12345)
lsav(x, z, u = diag(100), lbd = 1, add = .01)
```

```
## itel     1 fold  378.2744 fnew  248.2812
## itel     2 fold  248.2812 fnew  242.2507
## itel     3 fold  242.2507 fnew  228.3797
## itel     4 fold  228.3797 fnew  215.7506
## itel     5 fold  215.7506 fnew  212.6966
## itel     6 fold  212.6966 fnew  210.7577
## itel     7 fold  210.7577 fnew  206.3166
## itel     8 fold  206.3166 fnew  204.9684
## itel     9 fold  204.9684 fnew  204.6495
## itel    10 fold  204.6495 fnew  204.3335
## itel    11 fold  204.3335 fnew  203.9977
## itel    12 fold  203.9977 fnew  203.8240
## itel    13 fold  203.8240 fnew  203.7877
## itel    14 fold  203.7877 fnew  203.7826
## itel    15 fold  203.7826 fnew  203.7820
## itel    16 fold  203.7820 fnew  203.7819
```

```
## $coef
##                [,1]
## [1,] -0.2235170501
## [2,]  0.4705989074
## [3,] -0.8189051625
##
## $fit
## [1] 203.7819617
```

```
lsav(x, z, u = diag(100) - 1/100, lbd = 1, add = .01)
```

```
## itel     1 fold  361.6177 fnew  273.8130
## itel     2 fold  273.8130 fnew  235.7611
## itel     3 fold  235.7611 fnew  218.7905
## itel     4 fold  218.7905 fnew  209.7559
## itel     5 fold  209.7559 fnew  203.4202
## itel     6 fold  203.4202 fnew  198.6272
## itel     7 fold  198.6272 fnew  196.0594
## itel     8 fold  196.0594 fnew  194.6956
## itel     9 fold  194.6956 fnew  193.6528
## itel    10 fold  193.6528 fnew  192.9150
## itel    11 fold  192.9150 fnew  192.4444
## itel    12 fold  192.4444 fnew  192.1502
## itel    13 fold  192.1502 fnew  191.9642
## itel    14 fold  191.9642 fnew  191.8443
## itel    15 fold  191.8443 fnew  191.7658
## itel    16 fold  191.7658 fnew  191.7138
## itel    17 fold  191.7138 fnew  191.6791
## itel    18 fold  191.6791 fnew  191.6560
## itel    19 fold  191.6560 fnew  191.6407
## itel    20 fold  191.6407 fnew  191.6306
## itel    21 fold  191.6306 fnew  191.6240
## itel    22 fold  191.6240 fnew  191.6197
## itel    23 fold  191.6197 fnew  191.6169
## itel    24 fold  191.6169 fnew  191.6151
## itel    25 fold  191.6151 fnew  191.6139
## itel    26 fold  191.6139 fnew  191.6131
## itel    27 fold  191.6131 fnew  191.6126
## itel    28 fold  191.6126 fnew  191.6123
## itel    29 fold  191.6123 fnew  191.6121
## itel    30 fold  191.6121 fnew  191.6120
## itel    31 fold  191.6120 fnew  191.6119
```

```
## $coef
```

```
##                 [,1]
## [1,] -0.07636611408
## [2,]  0.26077579119
## [3,] -0.45976021741
##
## $fit
## [1] 191.6119645
```

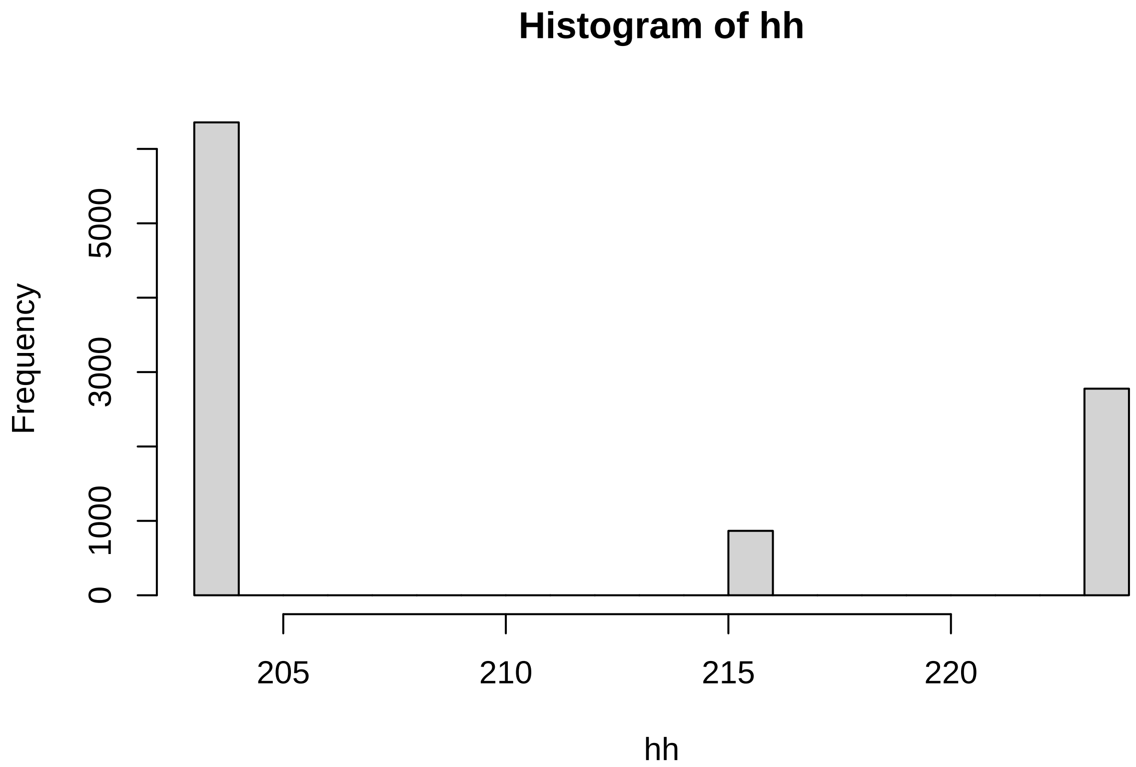
```
lsav(x, z, u = matrix (1/100, 100, 100), lbd = 1, add = .01)
```

```
## itel     1 fold   16.6567 fnew    2.9539
## itel     2 fold    2.9539 fnew    0.5309
## itel     3 fold    0.5309 fnew    0.0961
## itel     4 fold    0.0961 fnew    0.0175
## itel     5 fold    0.0175 fnew    0.0032
## itel     6 fold    0.0032 fnew    0.0006
## itel     7 fold    0.0006 fnew    0.0001
## itel     8 fold    0.0001 fnew    0.0000
```

```
## $coef
##               [,1]
## [1,] 0.6938729954
## [2,] 0.7085052814
## [3,] 0.7131573295
##
## $fit
## [1] 0.0001052784261
```

The example gives some indication that using a smoothing parameter decreases the frequency of local minima.

```
set.seed(12345)
hh <- c()
for (i in 1:10000) {
  h <- lsav(x, z, aold = rnorm (3) ^ 2, u = diag(100), lbd = 1, verbose = FALSE, add = .
  hh <- c(hh, h$fit)
}
hist(hh)
```



6 Appendix: A Cone Projection Result

The result in this appendix is there purely for historical reasons. Just ignore it.

Theorem: If $\hat{\beta}$ minimizes f and $z'U|X\hat{\beta}| > 0$ then

$$\tilde{\beta} = \frac{\hat{\beta}}{\sqrt{|X\hat{\beta}|'U|X\hat{\beta}|}}$$

maximizes $z'U|X\beta|$ over $|X\beta|'U|X\beta| = 1$ (and over $|X\beta|'U|X\beta| \leq 1$).

Proof:

$$\min_{\beta} (z - |X\beta|)'U(z - |X\beta|) = \min_{|\beta' X|'U|X\beta|=1} \min_{\gamma\geq 0} (z - \gamma|X\beta|)'U(z - \gamma|X\beta|).$$

From

$$\min_{\gamma\geq 0} (z - \gamma|X\beta|)'U(z - \gamma|X\beta|) = \begin{cases} z'Uz & \text{if } z'U|X\beta| < 0, \\ z'Uz - (z'U|X\beta|)^2 & \text{if } z'U|X\beta| \geq 0, \end{cases}$$

we get the statement in the theorem. ■

7 Appendix: Code

```
myAbs <- function (x, add = 0) {
  return (sqrt (x ^ 2 + add))
}

f <- function (alpha,
               x,
               z,
               u,
               add = 0.0) {
  r <- z - myAbs(x %*% alpha, add)
  return (sum (r * (u %*% r)))
}


lsav <-
  function (x,
            z,
            u,
            lbd = max(eigen(u)$values),
            aold = rep(1, ncol(x)),
            itmax = 100,
            eps = 1e-4,
            add = 0.0,
            verbose = TRUE) {
    n <- nrow (x)
    p <- ncol (x)
    v <- drop (u %*% z)
    vp <- ifelse (v >= 0, v, 0)
    vm <- ifelse (v < 0, -v, 0)
    fold <- f(aold, x, z, u, add)
    itel <- 1
    repeat {
      h <- drop (x %*% aold)
      y <- myAbs (h, add)
      s <- h / y
      w <- drop ((u - lbd * diag (n)) %*% y)
      wp <- ifelse (w >= 0, w, 0)
      wm <- ifelse (w < 0, -w, 0)
      d <- (vm + wp) / y
      e <- (vp + wm) * s
      cp <-
```

```
        crossprod (x, (lbd + d) * x)
anew <- solve (cp, crossprod (x, e))
fnew <- f (anew, x, z, u, add)
if (verbose) {
  cat(
    "itel ",
    formatC(itel, digits = 3, format = "d"),
    "fold ",
    formatC(
      fold,
      digits = 4,
      width = 8,
      format = "f"
    ),
    "fnew ",
    formatC(
      fnew,
      digits = 4,
      width = 8,
      format = "f"
    ),
    "\n"
  )
}
if (((fold - fnew) < eps) || (itel == itmax))
  break
aold <- anew
fold <- fnew
itel <- itel + 1
    }
    return (list(
      coef = anew,
      fit = sum ((z - y) * (u %*% (z - y)))
    ))
  }
```

References

De Leeuw, J. 1984. "Differentiability of Kruskal's Stress at a Local Minimum." *Psychometrika* 49: 111–13.

———. 1994. "Block Relaxation Algorithms in Statistics." In *Information Systems and Data Analysis*, edited by H. H. Bock, W. Lenski, and M. M. Richter, 308–24. Berlin: Springer Verlag.

———. 2005. “Unidimensional Scaling.” In *The Encyclopedia of Statistics in Behavioral Science*, edited by B. S. Everitt and D. C, 4:2095–97. New York, N.Y.: Wiley.

———. 2018. “MM Algorithms for Smoothed Absolute Values.” 2018.

———. 2020. “Faster Multivariate Cumulants.” 2020.

Dielman, T. E. 2005. “Least Absolute Value Regression: Recent Contributions.” *Journal of Statistical Computation and Simulation* 75 (4): 263–86.

Lange, K. 2016 (in press). *MM Optimization Algorithms*.