Least Squares Solutions of Linear Inequality Systems

Jan de Leeuw

Version 21, December 20, 2016

Abstract

We discuss the problem of finding an approximate solution to an overdetermined system of linear inequalities, or an exact solution if the system is consistent. Theory and R code is provided for four different algorithms. Two techniques use active set methods for non-negatively constrained least squares, one uses alternating least squares, and one uses a nonsmooth Newton method.

Contents

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

1 Introduction

For another project (De Leeuw (2016a)) I needed a quick way to check if a system of inhomogeneous linear inequalities $Ax \leq b$ is consistent (i.e. has at least one solution). There is a substantial literature on this problem, but not much R code.

The approach we take here is to minimize the least squares loss function

$$\sigma(x,y) = \mathbf{SSQ}(Ax - b + y) = \frac{1}{2}\mathbf{SSQ}(\begin{bmatrix} A & I \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} - b), \tag{1}$$

with **SSQ** the sum of squares, which we minimize over x and over the *slack variables* $y \geq 0$. For brevity, we will call this problem $\mathcal{P}$. We call a solution (x, y) of $\mathcal{P}$ the *augmented least squares solution of the system of linear inequalities*.

For homogeneous linear inequalities least squares solutions were already computed in De Leeuw (1968) and De Leeuw (1969), in some shaky papers in which *alternating least squares* was first introduced as a general principle of algorithm construction. For homogeneous inequalities there is always a (trivial) solution with both $x = 0$ and $y = 0$, so an additional constraint such as $\|x\| = 1$ or $\|Ax = 1\|$ or $\|y\| = 1$ was used in these early papers.

Clearly σ of (1) is a convex quadratic in (x, y), which is bounded below, and by the Frank-Wolfe theorem (Frank and Wolfe (1956)) attains its unique minimum at some x and $y \geq 0$. The minimum is equal to zero if and only if the system is consistent. The matrix $\begin{bmatrix} A & -I \end{bmatrix}$ is singular, however, which means the minimizer is not necessarily unique.

Initially, there are no constraints on the shape of A, it can be singular, it can be tall, it can be wide. But we can do some preprocessing and replace A by an orthonormal basis for its column space. Thus we suppose without loss of generality from now on that A is an $n \times m$ matrix satisfying $A'A = I$, which eliminates at least one obvious type of non-uniqueness.

We analyze $\mathcal{P}$, using theorem 31.4 of Rockafellar (1970). The set of all $(x, y) \in \mathbb{R}^m \otimes \mathbb{R}^n$ with $y \geq 0$ is a closed convex cone $\mathcal{K}$. The negative of the polar of $\mathcal{K}$ is the set $\mathcal{K}^\star = \{(u, v) \mid x'u + y'v \geq 0\}$ for all $(x, y) \in \mathcal{K}$, and thus $\mathcal{K}^\star = \{(0, v) \mid v \geq 0\}$. The derivatives of σ are

$$\begin{bmatrix} \mathcal{D}_1\sigma(x,y) \\ \mathcal{D}_2\sigma(x,y) \end{bmatrix} = \begin{bmatrix} A' \\ I \end{bmatrix} (Ax + y - b) = \begin{bmatrix} x - A'(b - y) \\ y - (b - Ax) \end{bmatrix}.$$

If follows that (x, y) is a solution of problem $\mathcal{P}$ if and only if

$$\begin{aligned} x &= A'(b - y), \\ y &\geq b - Ax, \\ y &\geq 0, \\ y'(y - (b - Ax)) &= 0. \end{aligned}$$

The last three conditions taken together are equivalent to $y = (b - Ax)_+ = \max(0, b - Ax)$.

2 Direct Attack

2.1 Alternating Least Squares

In block relaxation algorithms, and specifically in alternating least squares (ALS) algorithms, we minimize over a subset of the variables while keeping all other variables fixed at their current values, and then we cycle over subsets (De Leeuw (1994)). In our case the two natural subsets are x and y, and the ALS algorithm is

$$\begin{aligned} x^{(k+1)} &= A'(b - y^{(k)}), \\ y^{(k+1)} &= \max(0, b - Ax^{(k+1)}). \end{aligned}$$

Because the loss function is strictly convex in x for given y and in y for given x the block relaxation algorithm converges linearly to a stationary value, which is necessarily a minimizer, and a fixed point of the ALS algorithm.

Convergence is very stable but typically slow, especially when the system is consistent, and consequently the minimum of (1) is zero. Iterations are very simple and fast however, and even for moderately sized problems, which take hundreds of iterations, they are still feasible. For historical and sentimental reasons I am fond of this approach, implemented in the R function `LinInEqALS`.

Two examples follow, the first for an inconsistent system, the second for a consistent one. We iterate until $\sigma^{(k)} - \sigma^{(k+1)} < 1e - 15$. The rate we compute is $(\sigma^{(k)} - \sigma^{(k+1)})/(\sigma^{(k-1)} - \sigma^{(k)})$. Since we ae only interested in the consistency of the system of inequalities we do not pay much attention to the solutions, which in the case of consistency are generally not unique anyway.

```
set.seed (12345)
a <- gsRC (matrix (rnorm (200), 100, 2))$x
b_inc <- rnorm (100)
print (LinInEqALS (a, b_inc, eps = 1e-15, itmax = 10000))
```

```
## $itel
## [1] 31
##
## $f
## [1] 43.98898673
##
## $rate
## [1] 0
##
## $x
##           x1           x2
```

```
## -2.102367012 -1.593688308
##
## $y
##   [1] 0.00000000000 0.00000000000 0.29226818673 0.77062801007 0.95642609281
##   [6] 0.00000000000 0.00000000000 0.84623837385 0.00000000000 0.00000000000
##  [11] 0.77639294539 0.00000000000 0.00000000000 2.36440166282 0.00000000000
##  [16] 0.37354250575 0.50187513940 0.00000000000 0.00000000000 0.10854865700
##  [21] 1.24583471840 0.00000000000 0.00000000000 0.00000000000 1.45367217040
##  [26] 0.79347163928 0.00000000000 0.30152165777 1.29967176293 0.00000000000
##  [31] 0.47740938316 0.00000000000 0.59718382916 0.35317372552 1.35670296207
##  [36] 0.00000000000 0.54249450483 0.00000000000 0.94762453833 0.18541647988
##  [41] 0.00000000000 0.00000000000 0.00000000000 0.00000000000 0.04086658402
##  [46] 1.10450073817 2.26171773960 0.14744417850 0.78500137979 0.55362324080
##  [51] 0.00000000000 1.41829948950 0.49622172135 0.00000000000 0.00000000000
##  [56] 0.64888753297 0.00000000000 0.00000000000 0.00000000000 0.00000000000
##  [61] 0.65858247403 1.08849768148 0.00000000000 0.00000000000 1.22558038930
##  [66] 0.00000000000 0.29291521318 0.55550868847 0.00000000000 0.00000000000
##  [71] 0.00000000000 0.00000000000 0.40487476850 0.00000000000 0.22737104575
##  [76] 1.38334449098 0.55016409530 0.00000000000 0.00000000000 0.35727445598
##  [81] 0.00000000000 0.33313970646 0.00000000000 2.19283572391 0.94092382350
##  [86] 0.68575050578 0.44767862221 0.00000000000 1.03573953197 0.81560753401
##  [91] 0.00000000000 0.00000000000 0.25446570183 0.38142357310 0.00000000000
##  [96] 0.42727190385 0.00000000000 0.00000000000 0.00000000000 0.34879884897
```

```
b_con <- rowSums (a) + rchisq (100, 1)
print (LinInEqALS (a, b_con, eps = 1e-15, itmax = 10000))
```

```
## $itel
## [1] 2207
##
## $f
## [1] 1.15281717e-13
##
## $rate
## [1] 0.9914436441
##
## $x
##           x1           x2
## 0.9965424624 0.9931101612
##
## $y
##   [1] 1.3710807606238 0.7112244915946 0.3051474985678 2.5977036417749
##   [5] 0.7444717439545 2.5821372546049 1.1287843969939 0.3869768458471
##   [9] 4.3749615787636 0.6062134580577 6.6367550530764 0.3960722003334
##  [13] 0.2248660184917 0.0188875092304 0.1359426626587 1.4833967581528
```

```
##  [17] 2.0181300013856 1.9067700509573 1.9707830529594 3.3980818422779
##  [21] 0.1913246758175 0.0030351558038 0.3050463852871 0.0155171015103
##  [25] 1.7136102751987 0.2501623770847 1.4618090262931 0.5367206990712
##  [29] 3.0478220930313 0.1950354566832 0.0039034945550 0.0243610856150
##  [33] 0.0019341019148 0.8021858024458 2.1919592200680 0.0363624499542
##  [37] 3.2712900858200 4.0261901696408 0.0089978729416 0.3012775906720
##  [41] 3.9281629811057 0.5797312292212 0.2020996906975 1.3485751076193
##  [45] 7.0435563094314 0.0384527747325 0.3184635169704 1.7398895099357
##  [49] 1.0216849445249 1.0212033569996 0.6630584136679 0.0044431326995
##  [53] 0.0344906906485 0.0686775214898 0.2256017240739 0.5777529356018
##  [57] 1.6061305801338 0.8111067653027 1.0462097365450 0.0524847577074
##  [61] 1.3921994947586 0.0076746138621 0.0868257455659 2.2271038815941
##  [65] 1.3589935661205 0.9491272346732 2.6965048191706 0.1385001939416
##  [69] 2.7117220157836 0.1179340716820 0.6487493500860 0.0348325975226
##  [73] 5.5394257159452 1.1976298437813 1.5197332406616 0.8206275900528
##  [77] 1.9979538850602 2.6502483949405 1.3418243347330 0.0913952890804
##  [81] 0.0000746808541 1.2568103008935 0.0641149494304 1.6295893112139
##  [85] 0.2271021191766 1.9909883749196 2.6135794710939 3.6732953161033
##  [89] 0.5372912633635 0.0853403626306 1.1404874564906 0.0270312726769
##  [93] 0.8875175112965 0.1001911031757 0.0700122613832 0.0017075796186
##  [97] 2.9454002940664 0.6686046863305 0.0000000000000 0.2681956482007
```

2.2 Quadratic Programming

Minimizing (1) over x and $y \geq 0$ is a quadratic programming problem, more precisely a partially non-negative least squares problem, in $n + m$ variables with a singular coefficient matrix $\begin{bmatrix} A & I \end{bmatrix}$. This is a somewhat unsatisfactory formulation, because it leaves the problem unnecessarily big and sparse, but it is easy to implement using the `pnnls` function from the `lsei` package (Wang, Lawson, and Hanson (2015)).

```
print (LinInEqPNNLS (a, b_inc))
```

```
## $f
## [1] 43.98898673
##
## $x
## [1] -2.102367021 -1.593688333
##
## $y
##   [1] 0.00000000000 0.00000000000 0.29226818774 0.77062800652 0.95642609345
##   [6] 0.00000000000 0.00000000000 0.84623837763 0.00000000000 0.00000000000
##  [11] 0.77639294224 0.00000000000 0.00000000000 2.36440166638 0.00000000000
##  [16] 0.37354250487 0.50187513885 0.00000000000 0.00000000000 0.10854865696
##  [21] 1.24583472265 0.00000000000 0.00000000000 0.00000000000 1.45367216858
```

```
##  [26] 0.79347163723 0.00000000000 0.30152166097 1.29967175789 0.00000000000
##  [31] 0.47740938767 0.00000000000 0.59718383146 0.35317372332 1.35670296886
##  [36] 0.00000000000 0.54249450213 0.00000000000 0.94762453950 0.18541647883
##  [41] 0.00000000000 0.00000000000 0.00000000000 0.00000000000 0.04086658447
##  [46] 1.10450074022 2.26171773503 0.14744418091 0.78500138232 0.55362324053
##  [51] 0.00000000000 1.41829949180 0.49622172186 0.00000000000 0.00000000000
##  [56] 0.64888753247 0.00000000000 0.00000000000 0.00000000000 0.00000000000
##  [61] 0.65858247287 1.08849768160 0.00000000000 0.00000000000 1.22558038652
##  [66] 0.00000000000 0.29291521520 0.55550868611 0.00000000000 0.00000000000
##  [71] 0.00000000000 0.00000000000 0.40487476809 0.00000000000 0.22737104215
##  [76] 1.38334449264 0.55016410193 0.00000000000 0.00000000000 0.35727445191
##  [81] 0.00000000000 0.33313970683 0.00000000000 2.19283572793 0.94092382601
##  [86] 0.68575050687 0.44767862619 0.00000000000 1.03573953180 0.81560753314
##  [91] 0.00000000000 0.00000000000 0.25446570471 0.38142357367 0.00000000000
##  [96] 0.42727190618 0.00000000000 0.00000000000 0.00000000000 0.34879884867
```

```
print (LinInEqPNNLS (a, b_con))
```

```
## $f
## [1] 0
##
## $x
## [1] 0.9689355493 1.0307665057
##
## $y
##   [1] 1.371878837700 0.717562144575 0.303252051954 2.601416978745
##   [5] 0.745638522266 2.579060449448 1.131722100255 0.380348716642
##   [9] 4.375850997823 0.602429610170 6.641057450434 0.406145209040
##  [13] 0.221159446127 0.015381884953 0.134143650341 1.487585751792
##  [17] 2.015842838518 1.903475175299 1.965721531021 3.399178452051
##  [21] 0.187696984944 0.002142429488 0.302029139376 0.005405874009
##  [25] 1.710718624121 0.259579513918 1.467402119181 0.534104826682
##  [29] 3.057526245713 0.199080352809 0.000000000000 0.027191669387
##  [33] 0.005700804511 0.811216414571 2.182680146012 0.041681060764
##  [37] 3.274186067308 4.019016895660 0.013464677337 0.302939678575
##  [41] 3.935614661693 0.577929161944 0.201366415437 1.356847056407
##  [45] 7.045890637770 0.040513206111 0.320325485282 1.738282720041
##  [49] 1.019947964525 1.017020226225 0.667636536866 0.007836968335
##  [53] 0.033906156909 0.072696970591 0.224130461108 0.579475798628
##  [57] 1.601673881524 0.814710158010 1.051964753219 0.048356696509
##  [61] 1.394453188706 0.002772643575 0.089286318779 2.233370952641
##  [65] 1.361080577659 0.945049174376 2.696611675712 0.147629174614
##  [69] 2.706532504236 0.114392980928 0.648150238907 0.027565841379
##  [73] 5.536349953368 1.201057208513 1.530069016536 0.821459135305
##  [77] 1.990925375648 2.646464645716 1.342349144747 0.101087612822
```

```
##  [81] 0.005313001878 1.259917466174 0.064787306304 1.632276943783
##  [85] 0.226755355341 1.995932068833 2.609993160349 3.671024999486
##  [89] 0.539437448476 0.089540863339 1.135742009369 0.026536112949
##  [93] 0.889844261729 0.097961096205 0.062335486300 0.000629497511
##  [97] 2.940784208763 0.676318898769 0.000000000000 0.266199630302
```

3 Using Projection

Projection means constructing the two new loss functions $\sigma_\star(x) = \min_{y\geq 0}\sigma(x,y)$ and $\sigma_\dagger(y) = \min_x \sigma(x,y)$, and then minimizing those projected loss functions. Note that $\sigma_\star$ is a function of m variables, while $\sigma_\dagger$ is a function of n variables.

3.1 Projecting out x

We have

$$\sigma_\dagger(y) := \min_x \sigma(x,y) = \mathbf{SSQ}(A(A'(b-y))-(b-y)) = \mathbf{SSQ}((I-AA')(b-y)).$$

Choose an orthonormal basis $A_\perp$ for the null space of A, and set $c = A'_\perp b$. Then

$$\sigma_\dagger(y) = \mathbf{SSQ}(A'_\perp y - c),$$

and we must minimize $\sigma_\dagger(y)$ over $y \geq 0$, which can be done with the `nnls` function of `lsei` (Wang, Lawson, and Hanson (2015)). For null space bases and least squares solutions our function `LinInEqNNLS` uses the routines of De Leeuw (2016b). We reanalyze our two examples in this way.

```
print (LinInEqNNLS (a, b_inc))
```

```
## $f
## [1] 43.98898673
##
## $x
## [1] -2.102367021 -1.593688333
##
## $y
##   [1] 0.00000000000 0.00000000000 0.29226818774 0.77062800652 0.95642609345
##   [6] 0.00000000000 0.00000000000 0.84623837763 0.00000000000 0.00000000000
##  [11] 0.77639294224 0.00000000000 0.00000000000 2.36440166638 0.00000000000
##  [16] 0.37354250487 0.50187513885 0.00000000000 0.00000000000 0.10854865696
##  [21] 1.24583472265 0.00000000000 0.00000000000 0.00000000000 1.45367216858
##  [26] 0.79347163723 0.00000000000 0.30152166097 1.29967175789 0.00000000000
##  [31] 0.47740938767 0.00000000000 0.59718383146 0.35317372332 1.35670296886
##  [36] 0.00000000000 0.54249450213 0.00000000000 0.94762453950 0.18541647883
```

```
##  [41] 0.00000000000 0.00000000000 0.00000000000 0.00000000000 0.04086658447
##  [46] 1.10450074022 2.26171773503 0.14744418091 0.78500138232 0.55362324053
##  [51] 0.00000000000 1.41829949180 0.49622172186 0.00000000000 0.00000000000
##  [56] 0.64888753247 0.00000000000 0.00000000000 0.00000000000 0.00000000000
##  [61] 0.65858247287 1.08849768160 0.00000000000 0.00000000000 1.22558038652
##  [66] 0.00000000000 0.29291521520 0.55550868611 0.00000000000 0.00000000000
##  [71] 0.00000000000 0.00000000000 0.40487476809 0.00000000000 0.22737104215
##  [76] 1.38334449264 0.55016410193 0.00000000000 0.00000000000 0.35727445191
##  [81] 0.00000000000 0.33313970683 0.00000000000 2.19283572793 0.94092382601
##  [86] 0.68575050687 0.44767862619 0.00000000000 1.03573953180 0.81560753314
##  [91] 0.00000000000 0.00000000000 0.25446570471 0.38142357367 0.00000000000
##  [96] 0.42727190618 0.00000000000 0.00000000000 0.00000000000 0.34879884867
```

```
print (LinInEqNNLS (a, b_con))
```

```
## $f
## [1] 0
##
## $x
## [1] 0.9689355493 1.0307665057
##
## $y
##   [1] 1.371878837700 0.717562144575 0.303252051954 2.601416978745
##   [5] 0.745638522266 2.579060449448 1.131722100255 0.380348716642
##   [9] 4.375850997823 0.602429610170 6.641057450434 0.406145209040
##  [13] 0.221159446127 0.015381884953 0.134143650341 1.487585751792
##  [17] 2.015842838518 1.903475175299 1.965721531021 3.399178452051
##  [21] 0.187696984944 0.002142429488 0.302029139376 0.005405874009
##  [25] 1.710718624121 0.259579513918 1.467402119181 0.534104826682
##  [29] 3.057526245713 0.199080352809 0.000000000000 0.027191669387
##  [33] 0.005700804511 0.811216414571 2.182680146012 0.041681060765
##  [37] 3.274186067308 4.019016895660 0.013464677337 0.302939678575
##  [41] 3.935614661693 0.577929161944 0.201366415437 1.356847056407
##  [45] 7.045890637770 0.040513206111 0.320325485282 1.738282720041
##  [49] 1.019947964525 1.017020226225 0.667636536866 0.007836968335
##  [53] 0.033906156909 0.072696970591 0.224130461108 0.579475798628
##  [57] 1.601673881524 0.814710158010 1.051964753219 0.048356696509
##  [61] 1.394453188706 0.002772643575 0.089286318779 2.233370952641
##  [65] 1.361080577659 0.945049174376 2.696611675712 0.147629174614
##  [69] 2.706532504236 0.114392980928 0.648150238907 0.027565841379
##  [73] 5.536349953368 1.201057208513 1.530069016536 0.821459135305
##  [77] 1.990925375648 2.646464645716 1.342349144747 0.101087612822
##  [81] 0.005313001878 1.259917466174 0.064787306304 1.632276943783
##  [85] 0.226755355341 1.995932068833 2.609993160349 3.671024999486
##  [89] 0.539437448476 0.089540863339 1.135742009369 0.026536112949
```

```
##  [93] 0.889844261729 0.097961096205 0.062335486300 0.000629497511
##  [97] 2.940784208763 0.676318898769 0.000000000000 0.266199630302
```

The approach works well, but it has the disadvantage we still have to solve a problem with n variables, and that consequently the size of the null space basis L can be prohibitive.

3.2 Projecting out y

Projecting out y leads to a function of m variables. We can write

$$\sigma_\star(x) = \sigma(x, (b - Ax)_+) = \mathbf{SSQ}((Ax - b)_+).$$

Thus $\sigma_{(\bullet,\star)_\star}$ is convex, piecewise quadratic, and differentiable, although generally not twice differentiable.

Since Han (1980) the solution x minimizing $\sigma_\star$ is known as the least squares solution of the system of linear inequalities. Han (1980) shows that the problem of minimizing $\sigma_\star$ is equivalent to the quadratic program of minimizing $\frac{1}{2}z'z$ over $Ax - b \leq z$ over (x, z).

Han also proposes an efficient algorithm to minimize $\sigma_\star$. Various variations have been proposed by Yang (1990), Spoonamore (1992), Bramley and Winnicka (1996), Carp, Popa, and Serban (2013), and Popa and Serban (2014), but we implement the original algorithm. What is basically the Han algorithm is presented as a generalized Newton method in Pinar (1998) and Sahahi and Ketabchi (2010).

Suppose $I(x) = \{i \mid a_i'x \leq b\}$, the index set of the violated inequalities. Then compute the *direction of descent* d.

$$d^{(k)} := \underset{d}{\mathbf{argmin}}\, \mathbf{SSQ}(A_{I^{(k)}}d - (b_{I^{(k)}} - A_{I^{(k)}}x^{(k)}))$$

If the least squares solution is not unique, we have to make a choice. Han (1980) used the minimum norm least squares solution, while Bramley and Winnicka (1996) used the least squares solution based on QR with pivoting. Since we have the code from De Leeuw (2016b), we use QR with pivoting as well. Next we compute the *step size* λ.

$$\lambda^{(k)} := \underset{\lambda}{\mathbf{argmin}}\, \sigma_\star(x^{(k)} + \lambda d^{(k)})$$

Since $\sigma_\star$ is a convex and piecewise quadratic function of λ we just move from left to right, going from one breakpoint to another, as in De Leeuw (2016a), until we hit the minimum corresponding with the smallest minimizer. To finish the iteration we make the step $x^{(k+1)} = x^{(k)} + \lambda^{(k)}d^{(k)}$. In our examples the Han algorithm finishes in three iterations, obviously with superlinear convergence rate.

```
print (LinInEqHan (a, b_inc))
```

```
## Iteration:    1 fold:   45.42077460 fnew:   43.98906283
## Iteration:    2 fold:   43.98906283 fnew:   43.98898673
```

```
## Iteration:    3 fold:   43.98898673 fnew:   43.98898673
## $x
## [1] -2.102367021 -1.593688333
##
## $f
## [1] 43.98898673
##
## $itel
## [1] 3
```

```
print (LinInEqHan (a, b_con))
```

```
## Iteration:    1 fold:    0.43919424 fnew:    0.00260193
## Iteration:    2 fold:    0.00260193 fnew:    0.00035001
## Iteration:    3 fold:    0.00035001 fnew:    0.00000000
## $x
## [1] 0.9914900477 1.0204472323
##
## $f
## [1] 2.582655301e-12
##
## $itel
## [1] 3
```

4 Discussion

We started this paper with the problem of minimizing the loss function $\sigma : \mathbb{R}^m \otimes \mathbb{R}^n \to \mathbb{R}_+$, and then define $\sigma_\star : \mathbb{R}^m \to \mathbb{R}_+$ and $\sigma_\dagger : \mathbb{R}^n \to \mathbb{R}_+$ by using projection. But we could also have started with the problem of minimizing $\sigma_\star$ on $\mathbb{R}^m$ and then use *augmentation* (De Leeuw (1994)) to arrive at minimizing σ over $\mathbb{R}^m \otimes \mathbb{R}^n_+$.

If the problem is to minimize a function f over X, then an augmentation of f is a function g on $X \otimes Y$ such that $f(x) = \min_{y \in Y} g(x, y)$ for all $x \in X$. Then augmented minimization problem is to minimize g over $x \in X$ and $y \in Y$. In our example $\sigma_\star = \min_{y \geq 0} \sigma(x, y)$ and thus σ is an augmentation of $\sigma_\star$.

There is little doubt that the Han algorithm is computationally superior to the othetr options, although it requires somewhat more intricate programming. The quadratic programming methods are very simple to implment, given the existence of `lsei` on CRAN, but they are wasteful because they do not use the special structure of the problem. The ALS method is cute and very easy to program from scratch, but it may be too slow for larger problems.

With different technology, and with more rigor and detail, there are similar arguments in a very nice recent expository paper by Hiriart-Urruty, Contesse, and Penot (accepted 2016).

5 Appendix: Code

5.1 LinInEqALS.R

```
LinInEqALS <-
  function (a,
            b,
            itmax = 1000,
            eps = 1e-6,
            verbose = FALSE) {
    a <- gsRC (a)$x
    n <- length (b)
    y <- rep (0, n)
    itel <- 1
    fold <- Inf
    cold <- Inf
    repeat {
      h <- lm.fit (a, b - y, singular.ok = TRUE)
      x <- h$coefficients
      fnew <- sum (h$residuals ^ 2)
      cnew <- fold - fnew
      rate <- cnew / cold
      y <- pmax (0, b - a %*% x)
      if (verbose)
        cat(
          "Iteration: ",
          formatC (itel, width = 3, format = "d"),
          "fold: ",
          formatC (
            fold,
            digits = 8,
            width = 12,
            format = "f"
          ),
          "fnew: ",
          formatC (
            fnew,
            digits = 8,
            width = 12,
            format = "f"
          ),
          "rate: ",
          formatC (
```

```
            rate,
            digits = 8,
            width = 12,
            format = "f"
          ),
          "\n"
        )
      if ((itel == itmax) | (fold - fnew < eps))
        break
      itel <- itel + 1
      fold <- fnew
      cold <- cnew
    }
    return (list (
      itel = itel,
      f = fnew,
      rate = rate,
      x = x,
      y = y
    ))
  }
```

5.2 LinInEqPNNLS.R

```
LinInEqPNNLS <- function (a, b) {
  h <- pnnls (cbind(a, diag (nrow (a))), b, k = ncol (a))
  return (list (
    f = h$rnorm ^ 2,
    x = h$x[1:ncol(a)],
    y = h$x[-(1:ncol(a))]
  ))
}
```

5.3 LinInEqNNLS.R

```
LinInEqNNLS <- function (a, b) {
  l <- nullRC (t (a))
  u <- drop (crossprod (l, b))
  h <- nnls (t(l), u)
  return (list (
    f = h$rnorm ^ 2,
```

```
    x = lsRC (a, b - h$x)$solution,
    y = h$x
  ))
}
```

5.4 LinInEqHan.R

```
LinInEqHan <-
  function (a,
            b,
            itmax = 1000,
            eps = 1e-10,
            verbose = TRUE) {
    xold <- lsRC (a, b)$solution
    fold <- sum (pmax (0, (a %*% xold) - b) ^ 2)
    itel <- 1
    repeat {
      u <- drop (a %*% xold)
      k <- which (u >= b)
      if (length (k) == 0)
        break
      r <- u - b
      d <- -lsRC (a[k, , drop = FALSE], r[k])$solution
      s <- drop (a %*% d)
      func <- function (p) {
        sum (pmax (0, p * s + r)  ^ 2)
      }
      noto <- which (s != 0)
      knot <- sort (-unique (r[noto] / s[noto]))
      grad <-
        sapply (knot, function (q)
          sum (s * pmax (0, q * s + r)))
      pos_grad <- which (grad > 0)
      if (length (pos_grad) == 0)
        lbd <- 0
      else {
        first_pos <- min (pos_grad)
        last_neg <- first_pos - 1
        lbd <-
          optimize (func, c (knot[last_neg], knot[first_pos]))$minimum
      }
      xnew <- xold + lbd * d
      fnew <- sum (pmax (0, (a %*% xnew) - b) ^ 2)
```

```
    if (verbose)
      cat(
        "Iteration: ",
        formatC (itel, width = 3, format = "d"),
        "fold: ",
        formatC (
          fold,
          digits = 8,
          width = 12,
          format = "f"
        ),
        "fnew: ",
        formatC (
          fnew,
          digits = 8,
          width = 12,
          format = "f"
        ),
        "\n"
      )
    if ((itel == itmax) || ((fold - fnew) < eps) || (fnew < eps))
      break
    itel <- itel + 1
    xold <- xnew
    fold <- fnew
  }
  return (list (x = xnew, f = fnew, itel = itel))
}
```

References

Bramley, R., and B. Winnicka. 1996. "Solving Linear Inequalities in a Least Squares Sense." *SIAM Journal Scientific Computing* 17 (1): 275–86.

Carp, D., C. Popa, and C. Serban. 2013. "Iterative Solution of Inconsistent Systems of Linear Inequalities." *Proceedings Applied Mathematics and Mechanics* 13: 407–8. doi:10.1002/pamm.201310199.

De Leeuw, J. 1968. "Nonmetric Discriminant Analysis." Research Note 06-68. Department of Data Theory, University of Leiden. http://www.stat.ucla.edu/~deleeuw/janspubs/1968/reports/deleeuw_R_68d.pdf.

———. 1969. "The Linear Nonmetric Model." Research Note 003-69. Department of Data Theory FSW/RUL. http://www.stat.ucla.edu/~deleeuw/janspubs/1969/reports/deleeuw_

R_69c.pdf.

———. 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. http://www.stat.ucla.edu/~deleeuw/janspubs/1994/chapters/deleeuw_C_94c.pdf.

———. 2016a. "Discrete Minimax by Quadratic Majorization." doi:10.13140/RG.2.2.27781.14565.

———. 2016b. "In Praise of QR." doi:10.13140/RG.2.1.4581.8641.

Frank, M., and P. Wolfe. 1956. "An Algorithm for Quadratic Programming." *Naval Research Logistics Quarterly* 3: 95–110.

Han, S.P. 1980. "Least-squares Solutions of Linear Inequalities." Technical Report TR-2141. Mathematics Research Center, University of Wisconsin-Madison.

Hiriart-Urruty, J.-B., Contesse L, and J.P. Penot. accepted 2016. "Least Squares Solutions of Linear Inequality Systems: a Pedestrian Approach." *RAIRO Operations Research*. doi:http://dx.doi.org/10.1051/ro/2016042.

Pinar, M.C. 1998. "Newton's Method for Linear Inequality Systems." *European Journal of Operational Research* 107: 710–19.

Popa, C., and C. Serban. 2014. "Han-type Algorithms for Inconsistent Systems of Linear Inequalities – A Unified Approach." *Applied Mathematics and Computation* 246: 247–56.

Rockafellar, R.T. 1970. *Convex Analysis*. Princeton University Press.

Sahahi, M., and S. Ketabchi. 2010. "Correcting an Inconsistent Set of Linear Inequalities by the Generalized Newton Method." *Optimization Methods and Software* 25 (3): 457–65.

Spoonamore, J.H. 1992. "Least Square Methods for Solving Systems of Inequalities with Applications to an Assignment Problem." USACERL Technical Manuscript FF-93/03. US Army Corps of Engineers, Construction Enigineering Research Laboratory.

Wang, Y., C.L. Lawson, and R.J. Hanson. 2015. *lsei: Solving Least Squares Problems under Equality/Inequality Constraints*. http://CRAN.R-project.org/package=lsei.

Yang, K. 1990. "New Iterative Methods for Linear Inequalities." Technical Report 90-6. Department of Industrial; Operations Engineering, University of Michigan.