ACTIVE SET METHODS
FOR ISOTONE OPTIMIZATION

JAN DE LEEUW

ABSTRACT. Isotone optimization is formulated as a convex programming problem with simple linear constraints. A R implementation of a particular active set strategy is discussed, and applied to various isotone optimization problems important in statistics. The implementation is user-extendable, and handles a great many convex loss functions and partial orders.

1. INTRODUCTION

Suppose $\mathscr{I}_n = \{1, 2, \cdots, n\}$ and $\succeq$ is a partial order on $\mathscr{I}_n$. A vector $x \in \mathbb{R}^n$ is $\succeq$-*isotone* if $x_i \geq x_j$ for all index pairs with $i \succeq j$. In this paper we study the problem $\mathscr{P}(f, \succeq)$ of minimizing a closed proper convex function $f : \mathbb{R}^n \Rightarrow \mathbb{R}$ over all $\succeq$-*isotone* vectors. Note that because x is totally ordered, all $\succeq$-*isotone* vectors define linear extensions of the partial order $\succeq$. To prevent various kinds of problems that are irrelevant for our purposes anyway, we assume that f is continuous and bounded below by zero.

The inequalities defining isotonicity can be written in matrix form as $Ax \geq 0$, where A is a matrix in which each row corresponds with an index pair (i, j) such that $i \succeq j$. Such a row has element i equal to $+1$, element j equal to -1, and the rest of the elements equal to zero.

In order to eliminate redundancies we include a row for a pair (i, j) if and only if i *covers* j, which means that $i \succeq j$ and there is no $k \neq i, j$ such that $i \succeq k \succeq j$. Thus the rows are taken from the *cover graph* (or the *Hasse diagram*) of the partial order. Figure 1 gives some examples.

$\boxed{\textit{Insert Figure } 1 \textit{ about here}}$

Date: September 30, 2008 — 19h 41min — Typeset in TIMES ROMAN.

2000 *Mathematics Subject Classification.* 00A00.

Key words and phrases. Binomials, Normals, LaTeX.

The isotone programming problem $\mathscr{P}(f,\succeq)$ can thus also be written as a convex programming problem $\mathscr{P}(f,A)$ with linear inequality constraints.

2. Conditions for a Minimum

2.1. **Kuhn-Tucker Vectors.** A convex function f is minimized on a convex set $\{x \mid Ax \geq 0\}$ at $\hat{x}$ if and only if there exist a vector of Lagrange multipliers $\hat{\lambda}$ (also known as a *Kuhn-Tucker vector*) such that [Rockafellar, 1970, Chapter 28]

$$A'\hat{\lambda} \in \partial f(\hat{x}), \quad A\hat{x} \geq 0, \quad \hat{\lambda} \geq 0, \quad \hat{\lambda}'A\hat{x} = 0.$$

Here $\partial f(\hat{x})$ is the *subdifferential* of f at $\hat{x}$. The subdifferential at x is the set of all *subgradients* of f at x, where y is a subgradient at x if

$$f(z) \geq f(x) + (z-x)'y \quad \forall z.$$

In general, the subdifferential is a convex compact set. If f is differentiable at x there is a unique subgradient, the gradient $\nabla f(x)$. Thus the necessary and sufficient conditions for a $\hat{x}$ to be a minimizer in the differentiable case are existence of a Kuhn-Tucker vector $\hat{\lambda}$ such that

$$\nabla f(\hat{x}) = A'\hat{\lambda}, \quad A\hat{x} \geq 0, \quad \hat{\lambda} \geq 0, \quad \hat{\lambda}'A\hat{x} = 0.$$

2.2. **Auxilary Problems.** We now define a number of related problems, all for a given f and a given $m \times n$ matrix A. Problem $\mathscr{P}$ is to minimize f over $Ax \geq 0$. The minimum is $\hat{f}$ and the minimizer is $\hat{x}$.

Write I for subsets of the index set $\mathscr{I} = \{1,2,\cdots,m\}$. Then $A(I)$ is the corresponding $\textbf{card}(I) \times n$ submatrix of A, and $A(\overline{I})$ is the $(m - \textbf{card}(I)) \times n$ complementary submatrix. The *active constraints* at x, which we write as $I(x)$, are the indices i for which $a_i'x = 0$.

Problem $\mathscr{P}_+(I)$ is to minimize f over $A(I)x = 0$ and $A(\overline{I})x > 0$. The solutions is $\hat{x}_+(I)$, and the minimum value is $\hat{f}_+(I) = f(\hat{x}_+(I))$. We have $\hat{f}_+(I)) \geq \hat{f}$ for all $I \subseteq I$. Because the partitioning into equality and strict inequality constraints partitions the constraint set $\{x \mid Ax \geq 0\}$ into 2^m faces, some of which may be empty, we also have $\hat{f} = \min_{I \subseteq \mathscr{I}} \hat{f}_+(I)$. Solution $\hat{x}_+(I)$ is optimal for $\mathscr{P}_+(I)$ if and only if there exist a Kuhn-Tucker vector $\hat{\lambda}_+(I)$ such that

$$A(I)'\hat{\lambda}_+(I) \in \partial f(\hat{x}_+(I)), \quad A(I)\hat{x}_+(I) = 0, \qquad A(\overline{I})\hat{x}_+(I) > 0.$$

If follows that if $\hat{\lambda}_+(I) \geq 0$ then actually $\hat{x}_+(I)$ is optimal for $\mathscr{P}$. Conversely, if $I(\hat{x})$ are the indices of the active constraints at the solution $\hat{x}$ of $\mathscr{P}$, then $\hat{x}$ also solves $\mathscr{P}_+(I(\hat{x}))$.

We also define the 2^m problems $\mathscr{P}(I)$, which is to minimize f over $A(I)x = 0$, with minimum value $\hat{f}(I)$ and solution $\hat{x}(I)$. Now $\hat{f}(I) \leq \hat{f}_+(I)$ and $\hat{f}(I) \leq \hat{f}$ for all $I \subseteq \mathscr{I}$. $\hat{x}(I)$ is a solution if and only if there exists a Kuhn-Tucker vector $\hat{\lambda}(I)$ such that

$$A(I)'\hat{\lambda}(I) \in \partial f(\hat{x}(I)), \quad A(I)\hat{x}(I) = 0.$$

It follows that if $A(\bar{I})\hat{x}(I) \geq 0$ and $\hat{\lambda}(I) \geq 0$ then $\hat{x}(I)$ solves problem $\mathscr{P}$. Conversely $\hat{x}$ solves $\mathscr{P}(I(\hat{x}))$ and $\hat{f} = \min_{I \subseteq \mathscr{I}} \hat{f}(I)$, with the minimum attained for $I(\hat{x})$. Thus if we knew $I(\hat{x})$ we could solve $\mathscr{P}$ by solving $\mathscr{P}(I)$.

Because the problems $\mathscr{P}(I)$ play an important part in the manifold suboptimization algorithm given below, we discuss an equivalent formulation. The constraints $A(I)x = 0$ define a relation $\approx_I$ on $\{1, 2, \cdots, n\}$, with $i \approx_I k$ if there is a row j of $A(I)$ in which both a_{ji} and a_{jk} are non-zero. The reflexive and transitive closure $\overline{\approx}_I$ of $\approx_I$ is an equivalence relation, which can be coded as an *indicator matrix* $G(I)$, i.e. a binary matrix in which all n rows have exactly one element equal to one. We can construct $G(I)$ from the adjacency matrix of $\overline{\approx}_I$ by selecting unique columns. Note that $G(I)$ is of full column-rank, even if $A(I)$ is singular. We write $r(I)$ for the number of equivalence classes of $\overline{\approx}_I$. Thus $G(I)$ is an $n \times r(I)$ matrix satisfying $A(I)G(I) = 0$, in fact $G(I)$ is a basis for the null space of $A(I)$. Moreover $D(I) \stackrel{\Delta}{=} G(I)'G(I)$ is diagonal and indicates the number of elements in each of the equivalence classes.

Problem $\mathscr{P}(I)$ can be written as minimization of $f(G(I)\xi)$ over $\xi \in \mathbb{R}^{r(I)}$, which is a convex unconstrained problem. A vector $\hat{\xi}(I)$ is a solution if and only if $0 \in G(I)'\partial f(G(I)\hat{\xi}(I))$. Then $\hat{x}(I) = G(I)\hat{\xi}$ solves $\mathscr{P}(I)$. If $0 \in G(I)'\partial f(G(I)\hat{\xi}(I))$ it follows that there is a non-empty intersection of the subgradient $\partial f(G(I)\hat{\xi}(I))$ and the row-space of $A(I)$, i.e. there is a Kuhn-Tucker vector $A(I)'\hat{\lambda}(I) \in \partial f(G(I)\hat{\xi}(I))$.

3. Algorithm

To solve the problem we use an *active set strategy*-[Gill et al., 1981, Chapter 5.2], in particular the *manifold optimization strategy* described in Zangwill [1967] and again in Zangwill [1969, Chapter 8]. We solve a finite sequence of subproblems

$\mathscr{P}(I)$, that minimize $f(x)$ over x satisfying $A(I)x$. After solving each of the problems we change the active set I, either by adding or by dropping a constraint. The algorithm can be expected be efficient if minimizing f under simple equality constraints, or equivalently minimizing $f(G\xi)$ over ξ, can be done quickly and reliably.

ST: Suppose $x^{(s-1)}$ is a candidate solution in iteration $s-1$. It defines the index sets $I^{(s-1)} = I(x^{(s-1)})$ and $\overline{I}^{(s-1)} = \mathscr{I} - I(x^{(s-1)})$.

EQ: Suppose $y^{(s-1)}$ is a solution of $\mathscr{P}(I^{(s-1)})$. Then $y^{(s-1)}$ can be either feasible or infeasible for $\mathscr{P}$, depending on if $A(\overline{I}^{(s-1)})y^{(s-1)} \geq 0$ or not.

IN: If $y^{(s-1)}$ is *infeasible* we choose $x^{(s)}$ on the line between $x^{(s-1)}$ and $y^{(s-1)}$, where it crosses the boundary of the feasible region. This defines a new and larger set of active constraints $I(x^{(s)})$. Go back to step **EQ**.

FS: If $y^{(s-1)}$ is *feasible* we determine the corresponding Lagrange multipliers $\lambda^{(s-1)}$ for $\mathscr{P}(I^{(s-1)})$. If $\lambda^{(s-1)} \geq 0$ we have solved $\mathscr{P}$. If $\min \lambda^{(s-1)} < 0$, we find the most negative Lagrange multiplier and drop the corresponding equality constraint from $I^{(s-1)}$ to define a new and smaller set of active constraints $I^{(s)}$. Go back to step **EQ**.

Convergence of the algorithm follows from the fact that f decreases in each step and an index set $\mathscr{I}$ is never repeated. For details we refer to the publications by Zangwill we mentioned above.

In step **IN** we solve

$$\max_{\alpha} x^{(s-1)} + \alpha(y^{(s-1)} - x^{(s-1)})$$
$$\text{over} \quad \min_{i \in A(\overline{I}^{(s-1)})} a_i'x^{(s-1)} + \alpha(a_i'y^{(s-1)} - a_i'x^{(s-1)}) \geq 0.$$

Finding the smallest Lagrange multiplier in step **FS** is straightforward to implement in the differentiable case. We have to solve $A(I^{(s-1)})'\lambda = \nabla f(y^{(s-1)})$, and because $G(I^{(s-1)})'\nabla f(y^{(s-1)}) = 0$ and $A(I^{(s-1)})$ is of full row-rank, there is a unique solution $\lambda^{(s-1)}$. In the convex non-differentiable case, which we discuss in the next section, matters are more complicated.

4. Some Nonsmooth Cases

If f is convex, but not differentiable, we have to deal with the fact that in general $\partial f(x)$ may not be a singleton. It is possible to develop a general theory for active set methods in this case [Panier, 1987], but we will just look at some important special cases.

4.1. **The ℓ_∞ (or weighted Chebyshev) norm.** In the ℓ_∞ case we must minimize

$$f(\xi) = \|h(\xi)\|_\infty = \max_{i=1}^{n} |w_i h_i(\xi)|,$$

where $h(\xi) = z - G\xi$ are the *residuals*. We assume, without loss of generality, that $w_i > 0$ for all i.

The minimization can be done for each of the r columns of the indicator matrix G separately, and the solution $\hat{\xi}_j$ is the corresponding *weighted mid-range*. More specifically, let $I(j) = \{i \mid g_{ij} = 1\}$. Then

$$f_j(\hat{\xi}_j) = \min_{\xi_j} \max_{i \in I(j)} |z_i - \xi_j| = \max_{i,k \in I(j)} \frac{w_i w_k}{w_i + w_k} |z_i - z_k|.$$

If the (not necessarily unique) maximum over $(i,k) \in I(j)$ is attained at $(i(j), k(j))$, then the minimum of f over ξ is attained at

$$\hat{\xi}_j = \frac{w_{i(j)} z_{i(j)} + w_{k(j)} z_{k(j)}}{w_{i(j)} + w_{k(j)}},$$

where we choose the order within the pair $(i(j), k(j))$ such that $z_{i(j)} \leq \hat{\xi}_j \leq z_{k(j)}$. Now

$$\min_{\xi} f(\xi) = \max_{j=1}^{r} f_j(\hat{\xi}_j).$$

These results also applies if $I(j)$ is a singleton $\{i\}$, in which case $\hat{\xi}_j = z_i$ and $f_j(\hat{\xi}_j) = 0$. Set $\hat{x} = G\hat{\xi}$.

Next we must compute a subgradient in $\partial f(\hat{x})$ orthogonal to G. Suppose the e_i is a unit weight vectors, i.e. a vector with all elements equal to zero, except element i which is equal to either plus or minus w_i. Consider the set $\mathscr{E}$ of the $2n$ unit weight vectors. Then $f(\xi) = \max_{e_i \in \mathscr{E}} e_i' h(\xi)$. Let $\mathscr{E}(\xi) = \{e_i \mid e_i' h(\xi) = f(\xi)\}$. Then, by the formula for the subdifferential of the pointwise maximum of a finite number of convex functions (also known as Danskin's Theorem [Danskin, 1966]), we have $\partial f(\xi) = \mathbf{conv}(\mathscr{E}(\xi))$, with $\mathbf{conv}()$ the convex hull.

Choose any j for which $f_j(\hat{\xi}_j)$ is maximal. Such a j may not be unique in general. The index pair $(i(j), k(j))$ corresponds with the two unit weight vectors with non-zero elements $-w_{i(j)}$ and $+w_{k(j)}$. The subgradient we choose is the convex combination which has element -1 at position $i(j)$ and element $+1$ at position

$k(j)$. It is orthogonal to G, and thus we can find a corresponding Kuhn-Tucker vector.

4.2. **The ℓ_1 (or weighted absolute value) norm.** For the ℓ_1 norm we find the optimum $\hat{\xi}$ by computing *weighted medians* instead of weighted mid-ranges. Uniqueness problems, and the subdifferentials, will generally be much less smaller than in the ℓ_∞. For ℓ_1 we define $\mathscr{E}$ to be the set of 2^n vectors $(\pm w_1, \pm w_2, \cdots, \pm w_n)$. The subdifferential is the convex hull of the vectors $e \in \mathscr{E}$ for which $e_i'h(\xi) = \min_\xi f(\xi)$. If $h_i(\xi) \neq 0$ then $e_i = \mathbf{sign}(h_i(\xi))w_i$, but if $h_i(\xi) = 0$ element e_i can be any number in $[-w_i, +w_i]$. Thus the subdifferential is a multidimensional rectangle. If the medians are not equal to the observations the loss function is differentiable. If $h_i(\xi) = 0$ for some i in $I(j)$ then we select the corresponding element in the subgradient in such a way that they add up to zero over all $i \in I(j)$.

5. Implementation Notes

5.1. **Computing Indicators.** We compute $G(I)$ from $A(I)$ in two steps. We first make the adjacency matrix of $\approx_I$ and add the identity to make it reflexive. We then apply Warshall's Algorithm [Warshall, 1962] to replace the adjacency matrix by that of the transitive closure $\overline{\approx}_I$, which is an equivalence relation. Thus the transitive adjacency matrix has blocks of ones for the equivalence classes. We then use the `unique()` function in `R` to select the unique rows of the transitive adjacency matrix, and transpose to get $G(I)$.

5.2. **Packaged Loss Functions.** There are three types of loss functions that actually are implemented in the package. Many more could be added with very little extra effort.

5.2.1. *Differentiable Convex Functions.* Since solving $\mathscr{P}(I)$ is an unconstrained convex problem, we can simply use the `optim()` in `R` to minimize $f(G(I)(\xi))$ over ξ, and then set $\hat{x} = G(I)\hat{\xi}$. We have implemented this for the differentiable case, using the `BFGS` option of `optim()`. This guarantees (if the optimum is found with sufficient precision) that the gradient at $\hat{x}$ is orthogonal to the indicator matrix $G(I)$, and consequently that Lagrange multipliers can be computed. By making sure that $A(I)$ has full row-rank, the Kuhn-Tucker vector is actually unique. The routine `fSolver()` takes the arguments `fobj()` and `gobj()`, which are functions returning the loss function value and the gradient. Because of the way `optim()` is written it is also possible to tackle problems with non-differentiable loss functions, or even non-convex ones. But we are not responsible if this gets you into trouble.

5.2.2. *Special Problems.* In some cases solving $\mathscr{P}(I)$, i.e. minimizing $f(G(I)(\xi))$ over ξ, can be done more efficiently because of the structure of the problem. This is true, in particular, for least squares, least absolute value, Chebyshev, and Quantile regression. The solvers are, respectively, `lsSolver()`, `dSolver()`, `mSolver()` and `pSolver()`. We have added `lfSolver()`, which solves $\mathscr{P}(I)$ for the more general least squares problem $f(x) = (z-x)'W(z-x)$, where W is a not necessarily diagonal positive semi-definite matrix of order n.

5.2.3. *Hybrids.* In addition we have some differentiable solvers which internally computes loss function value and gradient, and then calls `fSolver()`. So they do not need to be given `fobj()` and `gobj()`. The first is `oSolver()`, which minimizes $f(x) = \sum_{i=1}^{n} w_i|z_i - x_i|^p$ for some $p > 1$. The solver `aSolver()` does asymmetric least squares, as in Efron [1991]. `eSolver()` minimizes the familiar ℓ_1 approximation $f(x) = \sum_{i=1}^{n} w_i\sqrt{(z_i - x_i)^2 + \varepsilon}$. `sSolver()` minimizes negative Poisson likelihood, `hSolver()` does Huber-loss as in Huber [1981], and `iSolver()` does SILF-loss [Chu et al., 2004]. With little extra effort various other fashionable SVM and lasso isotone regressions could be added.

5.3. **User-defined Functions.** Since the driver function `activeSet()` has a separate `R` function to solve $\mathscr{P}(I)$ as one of its parameters, users can implement their own isotone regression methods. For differentiable convex function they can use `optim()`, or they can write their own subroutines. Note that it is not at all necessary that the problems are of the regression or projection type, i.e. minimize some norm $\|z - x\|$. In fact, the driver can be easily modified to deal with general convex optimization problems with linear inequality constraints which are not necessarily of the isotone type.

5.4. **Computing Lagrange Multipliers.** In each step of the algorithm we have selected a subgradient such that $A(I)'\lambda = \nabla f(x(I))$ is solvable, because we have made sure that $G(I)'\nabla f(x(I)) = 0$. In `R` we use `qr.coef(qr()` to actually compute the Kuhn-Tucker vector λ. By making sure that $A(I)$ has full row-rank, this Kuhn-Tucker vector is actually unique (for a given choice of the subgradient).

REFERENCES

W. Chu, S.S. Keerthi, and C.J. Ong. Bayesian Support Vector Regression Using a Unified Loss Function. *IEEE Transactions on Neural Networks*, 15:29–44, 2004.

J.M. Danskin. The Theory of Max-Min, with Applications. *SIAM Journal on Applied Mathematics*, 14:641–664, 1966.

B. Efron. Regression Percentiles using Asymmetric Squared Error Loss. *Statistica Sinica*, 1:93–125, 1991.

P.E. Gill, W. Murray, and M.H. Wright. *Practical Optimization*. Academic Press, New York, N.Y., 1981.

P. Huber. *Robust Regression*. Wiley, New York, NY, 1981.

E.R. Panier. An Active Set Method for Solving Linearly Constrained Nonsmooth Optimization Problems. *Mathematical Programming*, 37:269–292, 1987.

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

S. Warshall. A Theorem on Boolean Matrices. *Journal of the Association of Computer Machinery*, 9:11–12, 1962.

W. I. Zangwill. *Nonlinear Programming: a Unified Approach*. Prentice-Hall, Englewood-Cliffs, N.J., 1969.

W.I. Zangwill. A Decomposable Nonlinear Programming Approach. *Operations Research*, 15:1068–1087, 1967.

Appendix A. Code

A.1. Programs.

```
#
#   activeSet package
#   Copyright (C) 2008  Jan de Leeuw <deleeuw@stat.ucla.edu>
#   UCLA Department of Statistics, Box 951554,
#   Los Angeles, CA 90095-1554
#
#   This program is free software; you can redistribute it
#   and/or modify it under the terms of the GNU General Public
#   License as published by the Free Software Foundation;
#   either version 2 of the License, or (at your option)
#   any later version.
#
#   This program is distributed in the hope that it will be
#   useful, but WITHOUT ANY WARRANTY; without even the implied
#   warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
#   PURPOSE.  See the GNU General Public License for more details.
#
#   You should have received a copy of the GNU General Public
#   License along with this program; if not, write to the
#   Free Software Foundation, Inc., 675 Mass Ave, Cambridge,
#   MA 02139, USA.
#
###################################################################

#
# version 0.0.1, 2008-09-24, initial
# version 0.0.2, 2008-09-24, squashed a buggy
# version 0.1.0, 2008-09-25, replaced null space algorithm
# version 0.1.1, 2008-09-25, fSolver now works for iso=TRUE
# version 0.2.0, 2008-09-25, added dSolver, pSolver, mSolver
# version 0.2.1, 2008-09-25, corrected weightedMidRange
# version 0.2.2, 2008-09-26, improved qSolver
# version 1.0.0, 2008-09-26, throw out all iso=FALSE stuff
# version 1.0.1, 2008-09-27, additional xSolvers
# version 1.0.2, 2008-09-27, many buggies squashed
```

```
# version 1.0.3, 2008-09-27, check for optimality
# version 1.0.4, 2008-09-27, other rule to add to active set
# version 1.0.5, 2008-09-27, added SILF loss
# version 1.0.6, 2008-09-28, corrected mSolver
#
# To do (maybe):
#
# -- bound constraints
# -- regression constraints, as in (x-Zb)'W(x-Zb) with AZb>=0
#

activeSet<-function(a,x,mySolver=lsSolver,ups=1e-12,check=FALSE
    ,...) {
extra<-list(...); n<-length(x)
xold<-x; ax<-aTx(a,xold)
ia<-is.active(ax,ups=ups)
repeat {
    if (length(ia)==0) aia<-NULL
        else aia<-a[ia,]
    yl<-mySolver(xold,aia,extra)
    y<-yl$y; lbd<-yl$lbd; fy<-yl$f; gy<-yl$gy
    ay<-aTx(a,y)
    iy<-which.min(ay); my<-ay[iy]
    if (length(lbd)==0) ml<-Inf
        else {
            il<-which.min(lbd)
            ml<-lbd[il]
            }
    if (is.pos(my,ups)) {
            if (is.pos(ml,ups)) break()
            xnew<-y; ax<-ay
            ia<-ia[-il]
            }
    else {
        k<-which((ax>0)&(ay<0))
        rat<--ay[k]/(ax[k]-ay[k])
        ir<-which.max(rat); alw<-rat[ir]
        xnew<-y+alw*(xold-y)
        ax<-aTx(a,xnew)
```

```
        ia<-sort(c(ia,k[ir]))
        }
    xold<-xnew
    }
lup<-rep(0,length(ay)); lup[ia]<-lbd; hl<-taTx(a,lup,n)
if (check) ck<-checkSol(y,gy,a,ay,hl,lup,ups)
    else ck<-NULL
return(list(x=y,lbd=lup,f=fy,ay=ay,hl=hl,gy=gy,ck=ck))
}

# least squares with diagonal weights

lsSolver<-function(x,a,extra) {
    w<-extra$w; z<-extra$z; n<-length(z)
    if (length(a)==0) return(list(y=z,l=0,f=0))
    if (is.vector(a)) a<-matrix(a,1,length(a))
    indi<-mkIndi(a,n)
    h<-crossprod(indi,w*indi); r<-drop(crossprod(indi,w*z))
    b<-solve(h,r); y<-drop(indi%*%b); gy<-2*w*(y-z)
    lbd<-mkLagrange(a,gy)
    f<-sum(w*(y-z)^2)
    return(list(y=y,lbd=lbd,f=f,gy=gy))
}

# least squares with non-diagonal weights

lfSolver<-function(x,a,extra) {
    w<-extra$w; z<-extra$z; n<-length(z)
    if (length(a)==0) return(list(y=z,l=0,f=0))
    if (is.vector(a)) a<-matrix(a,1,length(a))
    indi<-mkIndi(a,n)
    h<-crossprod(indi,w%*%indi); r<-drop(crossprod(indi,w%*%z))
    b<-solve(h,r); y<-drop(indi%*%b); gy<-2*drop(w%*%(y-z))
    lbd<-mkLagrange(a,gy)
    f<-sum(w*outer(y-z,y-z))
    return(list(y=y,lbd=lbd,f=f,gy=gy))
}

# least absolute value
```

```
dSolver<-function(x,a,extra) {
    w<-extra$w; z<-extra$z; n<-length(z)
    if (length(a)==0) return(list(y=z,l=0,f=0))
    if (is.vector(a)) a<-matrix(a,1,2)
    indi<-mkIndi(a,n)
    m<-ncol(indi); h<-rep(0,m)
    for (j in 1:m) {
        ij<-which(indi[,j]==1)
        zj<-z[ij]; wj<-w[ij]
        h[j]<-weightedMedian(zj,wj)
        }
    y<-drop(indi%*%h); f<-sum(w*abs(z-y)); gy<-w*sign(y-z)
    lbd<-mkLagrange(a,gy)
    return(list(y=y,lbd=lbd,f=f,gy=gy))
}

# quantile loss function

pSolver<-function(x,a,extra) {
    w<-extra$w; z<-extra$z; aw<-extra$aw; bw<-extra$bw; n
        <-length(z)
    if (length(a)==0) return(list(y=z,l=0,f=0))
    if (is.vector(a)) a<-matrix(a,1,2)
    indi<-mkIndi(a,n)
    m<-ncol(indi); h<-rep(0,m)
    for (j in 1:m) {
        ij<-which(indi[,j]==1)
        zj<-z[ij]; wj<-w[ij]
        h[j]<-weightedFractile(zj,wj,aw,bw)
        }
    y<-drop(indi%*%h); dv<-ifelse(y<=z,w*aw*(z-y),w*bw*(y-z))
    f<-sum(dv); gy<-ifelse(y<=z,-w*aw,w*bw)
    lbd<-mkLagrange(a,gy)
    return(list(y=y,lbd=lbd,f=f,gy=gy))
}

# Chebyshev norm

```

```
mSolver<-function(x,a,extra) {
    w<-extra$w; z<-extra$z; n<-length(z)
    if (length(a)==0) return(list(y=z,l=0,f=0))
    if (is.vector(a)) a<-matrix(a,1,2)
    indi<-mkIndi(a,n)
    m<-ncol(indi); h<-rep(0,m)
    for (j in 1:m) {
        ij<-which(indi[,j]==1)
        zj<-z[ij]; wj<-w[ij]
        h[j]<-weightedMidRange(zj,wj)
        }
    y<-drop(indi%*%h); dv<-w*(y-z)
    i1<-which.max(dv); i2<-which.min(dv)
    f<-max(abs(dv))
    gy1<-rep(0,n); gy1[i1]<-w[i1]
    lbd1<-mkLagrange(a,gy1)
    gy2<-rep(0,n); gy2[i2]<--w[i2]
    lbd2<-mkLagrange(a,gy2)
    lbd<-(w[i2]*lbd1+w[i1]*lbd2)/(w[i1]+w[i2])
    gy<-(w[i2]*gy1+w[i1]*gy2)/(w[i1]+w[i2])
    return(list(y=y,lbd=lbd,f=f,gy=gy))
}

# arbitrary differentiable function

fSolver<-function(x,a,extra) {
    fobj<-extra$fobj; gobj<-extra$gobj; n<-length(x)
    if (length(a)==0) indi<-diag(n)
        else {
            if (is.vector(a)) a<-matrix(a,1,2)
            indi<-mkIndi(a,n)
            }
    z<-drop(crossprod(indi,x))
    p<-optim(z,
        fn=function(u) fobj(drop(indi%*%u)),
        gr=function(u) drop(crossprod(indi,gobj(drop(indi%*%u))
            )),
        method="BFGS")
    y<-drop(indi%*%(p$par)); f<-p$value; gy<-gobj(y)
```

```
    if (length(a)==0) lbd<-0
        else lbd<-mkLagrange(a,gy)
    return(list(y=y,lbd=lbd,f=f,gy=gy))
}

# Power Norms

oSolver<-function(x,a,extra) {
    w<-extra$w; z<-extra$z; pow<-extra$p
    fobj<-function(x) sum(w*(abs(x-z)^pow))
    gobj<-function(x) pow*w*sign(x-z)*abs(x-z)^(pow-1)
    return(fSolver(x,a,list(fobj=fobj,gobj=gobj)))
}

# Asymmetric Least Squares

aSolver<-function(x,a,extra) {
    w<-extra$w; z<-extra$z; aw<-extra$aw; bw<-extra$bw
    fobj<-function(x) sum(w*(x-z)^2*ifelse(x<z,aw,bw))
    gobj<-function(x) 2*w*(x-z)*ifelse(x<z,aw,bw)
    return(fSolver(x,a,list(fobj=fobj,gobj=gobj)))
}

# Approximate l_1

eSolver<-function(x,a,extra) {
    w<-extra$w; z<-extra$z; eps<-extra$eps
    fobj<-function(x) sum(w*sqrt((x-z)^2+eps))
    gobj<-function(x) w*(x-z)/sqrt((x-z)^2+eps)
    return(fSolver(x,a,list(fobj=fobj,gobj=gobj)))
}

# Poisson Likelihood

sSolver<-function(x,a,extra) {
    z<-extra$z
    fobj<-function(x) sum(x-z*log(x))
    gobj<-function(x) 1-z/x
    return(fSolver(x,a,list(fobj=fobj,gobj=gobj)))
```

```
}

# Huber Loss

hSolver<-function(x,a,extra) {
    w<-extra$w; z<-extra$z; eps<-extra$eps
    fobj<-function(x) sum(w*ifelse(abs(x-z)<2*eps,((x-z)^2)/(4
        *eps),abs(x-z)-eps))
    gobj<-function(x) w*ifelse(abs(x-z)<2*eps,((x-z))/(2*eps),
        sign(x-z))
    return(fSolver(x,a,list(fobj=fobj,gobj=gobj)))
}

# SILF Loss

iSolver<-function(x,a,extra) {
w<-extra$w; z<-extra$z; eps<-extra$eps; beta<-extra$beta
fobj<-function(x) {
    y<-abs(x-z)
    g<-((y-(1-beta)*eps)^2)/(4*beta*eps)
    g[which(y < (1-beta)*eps)]<-0
    ii<-which(y > (1+beta)*eps)
    g[ii]<-y[ii]-eps
    return(sum(w*g))
    }
gobj<-function(x) {
    y<-x-z
    g<-rep(0,length(y))
    g[which(y < -(1+beta)*eps)]<--1
    ii<-which((y > -(1+beta)*eps) & (y < -(1-beta)*eps))
    g[ii]<-(y[ii]+(1-beta)*eps)/(2*beta*eps)
    ii<-which((y > (1-beta)*eps) & (y < (1+beta)*eps))
    g[ii]<-(y[ii]-(1-beta)*eps)/(2*beta*eps)
    g[which(y > (1+beta)*eps)]<-1
    return(w*g)
    }
return(fSolver(x,a,list(fobj=fobj,gobj=gobj)))
}
```

```
aTx<-function(a,x) {
    if (is.vector(x)) return(x[a[,1]]-x[a[,2]])
    return(x[a[,1],]-x[a[,2],])
}

xT<-function(x) {
   if (is.vector(x)) return(as.matrix(x))
        else return(t(x))
}

taTx<-function(a,x,n) {
m<-nrow(a); h<-rep(0,n)
for (i in 1:m) {
    h[a[i,1]]<-h[a[i,1]]+x[i]
    h[a[i,2]]<-h[a[i,2]]-x[i]
    }
return(h)
}

b2a<-function(b,n) {
m<-nrow(b)
a<-matrix(0,m,n)
for (i in 1:m) {
    a[i,b[i,1]]<-1
    a[i,b[i,2]]<--1
    }
return(a)
}

warshall<-function(a) {
n<-nrow(a)
for (j in 1:n) {
    for(i in 1:n) {
        if (a[i,j]==1) a[i,]<-pmax(a[i,],a[j,])
        }
    }
return(a)
}
```

```
mkIndi<-function(a,n) {
im<-matrix(0,n,n); m<-nrow(a)
for (i in 1:m) im[a[i,1],a[i,2]]<-im[a[i,2],a[i,1]]<-1
im<-im+diag(n)
return(t(unique(warshall(im))))
}

mkLagrange<-function(b,g) {
ta<-t(b2a(b,length(g)))
qa<-qr(ta)
return(qr.coef(qr(ta),g))
}

checkSol<-function(y,gy,a,ay,hl,lbd,ups) {
ckFeasibility<-min(ay)
ckLagrange<-min(lbd)
ckCompSlack<-sum(ay*lbd)
ckGrad<-max(abs(gy-hl))
return(c(ckFeasibility,ckLagrange,ckCompSlack,ckGrad))
}

weightedMedian<-function(x,w=rep(1,length(x))){
ox<-order(x); x<-x[ox]; w<-w[ox]; k<-1
low<-cumsum(c(0,w)); up<-sum(w)-low; df<-low-up
repeat{
    if (df[k] < 0) k<-k+1
        else if (df[k] == 0) return((w[k]*x[k]+w[k-1]*x[k-1])/(
            w[k]+w[k-1]))
            else return(x[k-1])
    }
}

weightedFractile<-function(x,w=rep(1,length(x)),a=1,b=1){
ox<-order(x); x<-x[ox]; w<-w[ox]; k<-1
low<-cumsum(c(0,w)); up<-sum(w)-low; df<-a*low-b*up
repeat{
    if (df[k] < 0) k<-k+1
        else if (df[k] == 0) return((w[k]*x[k]+w[k-1]*x[k-1])/(
            w[k]+w[k-1]))
```

```
            else return(x[k-1])
    }
}

weightedMidRange<-function(x,w=rep(1,length(x))){
s<-0; n<-length(x)
if (n==1) return(x)
for (i in 1:(n-1)) for(j in (i+1):n) {
    t<-w[i]*w[j]*abs(x[i]-x[j])/(w[i]+w[j])
    if (t > s) {
        s<-t; i0<-i; j0<-j
        }
    }
return((w[i0]*x[i0]+w[j0]*x[j0])/(w[i0]+w[j0]))
}

is.active<-function(f,ups=1e-12) which(abs(f) < ups)

is.pos<-function(x,ups=1e-12) x > -ups

is.neg<-function(x,ups=1e-12) x < ups
```

A.2. **Examples.**

```
set.seed(12345)
z<-rnorm(9)
wu<-rep(1,9)
ww<-1:9
wf<-crossprod(matrix(rnorm(81),9,9))/9
x0<-9:1
btota<-cbind(1:8,2:9)
btree<-matrix(c(1,1,2,2,2,3,3,8,2,3,4,5,6,7,8,9),8,2)
bprim<-cbind(
    c(rep(1,3),rep(2,3),rep(3,3),rep(4,3),rep(5,3),rep(6,3)),
    c(rep(c(4,5,6),3),rep(c(7,8,9),3)))
bloop<-matrix(c(1,2,3,3,4,5,6,6,7,8,3,3,4,5,6,6,7,8,9,9),10,2)

compPava<-function(){
    cat("Comparison with gpava\n")
    library(pava)
```

```
    for (i in 1:100) {
        z<-rnorm(9)
        u<-gpava(x0,z)$yfit
        h<-activeSet(btota,x0,lsSolver,w=wu,z=z)$x
        k<-activeSet(btota,x0,fSolver,fobj=function(x) sum(wu*(
            x-z)^2),gobj=function(x) 2*drop(wu*(x-z)))$x
        print(max(apply(cbind(u,h,k),1,var)))
        }
}

otherWeights<-function(){
    cat("Diagonal Weights\n")
    print(activeSet(btota,x0,lsSolver,check=TRUE,w=ww,z=z))
    cat("Nondiagonal weights\n")
    print(activeSet(btota,x0,lfSolver,check=TRUE,x0,w=wf,z=z))
}

otherNorms<-function(){
    cat("Approximate l_1 with eps\n")
    print(activeSet(btota,x0,eSolver,check=TRUE,z=z,w=wu,eps=1e
        -4))
    cat("Approximate l_1 with power\n")
    print(activeSet(btota,x0,oSolver,check=TRUE,z=z,w=wu,p=1.2)
        )
    cat("Exact l_1\n")
    print(activeSet(btota,x0,dSolver,check=TRUE,w=wu,z=z))
    cat("Approximate l_infty with power\n")
    print(activeSet(btota,x0,oSolver,check=TRUE,z=z,w=wu,p=7))
    cat("Exact l_infty\n")
    print(activeSet(btota,x0,mSolver,check=TRUE,w=wu,z=z))
    cat("Poisson likelihood\n")
    z<-rpois(9,5)
    print(activeSet(btota,x0,sSolver,check=TRUE,w=wu,z=z))
    cat("Asymmetric Least Squares\n")
    print(activeSet(btota,x0,aSolver,check=TRUE,z=z,w=wu,aw=2,
        bw=1))
    cat("Huber Norm\n")
    print(activeSet(btota,x0,hSolver,check=TRUE,z=z,w=wu,eps=1)
        )
```

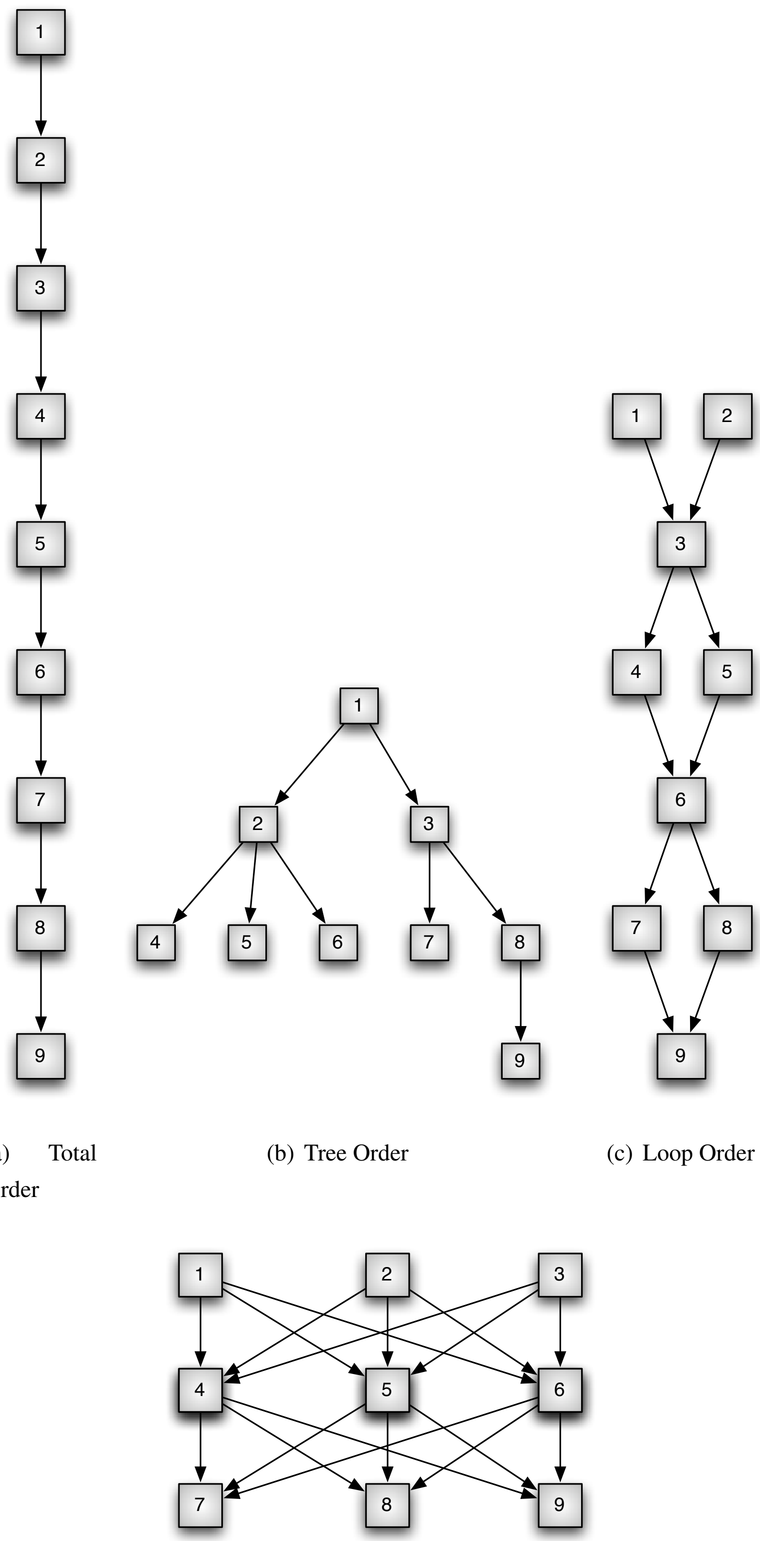
```
    cat("SILF Norm\n")
    print(activeSet(btota,x0,iSolver,check=TRUE,z=z,w=wu,beta=
        .8,eps=.2))
}

otherOrders<-function() {
    cat("Tree Order\n")
    print(activeSet(btree,x0,lsSolver,check=TRUE,w=wu,z=z))
    cat("Block Order\n")
    print(activeSet(bprim,x0,lsSolver,check=TRUE,w=wu,z=z))
    cat("Loop Order\n")
    print(activeSet(bloop,x0,lsSolver,check=TRUE,w=wu,z=z))
}
```

DEPARTMENT OF STATISTICS, UNIVERSITY OF CALIFORNIA, LOS ANGELES, CA 90095-1554

E-mail address, Jan de Leeuw: deleeuw@stat.ucla.edu

URL, Jan de Leeuw: http://gifi.stat.ucla.edu



(a) Total Order (b) Tree Order (c) Loop Order

(d) Block Order

FIGURE 1. Some Partial Orders