Bounded-Variable Least-Squares: an Algorithm and Applications

By

P.B. Stark
Department of Statistics
University of California
Berkeley, CA

and

R.L. Parker
Institute for Geophysics and Planetary Physics
University of California
San Diego, CA

Technical Report No. 394
July 1993

Department of Statistics
University of California
Berkeley, California 94720

Bounded-Variable Least-Squares: an Algorithm and Applications

P.B. Stark

Department of Statistics

University of California

Berkeley CA

R.L. Parker

Institute for Geophysics and Planetary Physics

University of California

San Diego CA

July 9, 1993

Abstract

The Fortran subroutine BVLS (bounded variable least-squares) solves linear least-squares problems with upper and lower bounds on the variables, using an active set strategy. The unconstrained least-squares problems for each candidate set of free variables are solved using the QR decomposition. BVLS has a "warm-start" feature permitting some of the variables to be initialized at their upper or lower bounds, which speeds the solution of a sequence of related problems. Such sequences of problems arise, for example, when BVLS is used to find bounds on linear functionals of a model constrained to satisfy, in an approximate l_p-norm sense, a set of linear equality constraints in addition to upper and lower bounds. We show how to use BVLS to solve that problem when $p = 1, 2$, or ∞, and to solve minimum l_1 and l_∞ fitting problems. We give computational examples using BVLS to find simultaneous confidence bounds

in monotone regression, which can be used to find confidence intervals in calibration problems.

Keywords: Optimization, constrained least-squares, l_1 and l_∞ regression, confidence envelopes, monotone regression.

Acknowledgments: The second author is supported by NSF PYI award DMS-8957573.

1 Introduction

In our research we have often encountered linear least-squares, l_1, and l_∞ regression problems with linear inequality constraints on the unknowns, as well as the problem of finding bounds on linear functionals subject to upper and lower bounds on the variables and a bound on the l_1, l_2, or l_∞ misfit to a set of linear relations ([4, 5, 10, 12, 11, 9, 3]). We have used the NNLS (non-negative least-squares) algorithm of Lawson and Hanson [2] to solve some of these problems, and in principle, NNLS can solve them all. However, in practice, when either (1) both lower and upper bounds on the variables are given, or (2) one must solve a sequence of related problems, NNLS can be impractical. In (1), the size of the matrix and the number of unknowns for NNLS is unnecessarily large, since many slack variables are needed. In (2), NNLS incurs high overhead in finding a good free set from scratch in each problem (it overwrites the information needed for warm starts). We have been able to solve much larger problems in much less time using an algorithm that explicitly incorporates upper and lower bounds on the variables and returns information about its final free and bound sets.

BVLS (bounded-variable least-squares) is modelled on NNLS and solves the problem *bvls*:

$$\min_{l \le x \le u} \|Ax - b\|_2 \tag{1}$$

where $l, x, u \in \mathbf{R}^n$, $b \in \mathbf{R}^m$, and A is an m by n matrix. The relative size of m and n is immaterial; typically, in inverse problems, $m << n$. Some commercial codes advertised to work when $m < n$, do not.

As we show below, if one can solve problem *bvls* one can also solve the problem *bvmm(p)*

(bounded-variable minimum misfit):

$$\min_{l \le x \le u} \|Ax - d\|_p \tag{2}$$

when $p = 1$ or $p = \infty$ ($p = 2$ is just *bvls*); as well as the problem *blf(p)*:

$$\min c \cdot x \tag{3}$$

subject to the constraints

$$l \le x \le u \tag{4}$$

and

$$\|Ax - d\|_p \le \chi \tag{5}$$

where p is any of $1, 2$, or ∞.

2 The BVLS Algorithm

BVLS uses an active set strategy similar to that of NNLS [2], except two active sets are maintained—one for variables at their lower bounds and one for variables at their upper bounds. The proof that BVLS converges to a solution of problem *bvls* follows that for NNLS [2]; we do not give it here.

Here is an outline of the algorithm, numbered to agree with Lawson and Hanson's notation. The set $\mathcal{F}$ contains the indices of "free" components of the working solution x that are strictly between their lower and upper bounds; $\mathcal{L}$ contains the indices of components at their lower bounds; and $\mathcal{U}$ contains the indices of components at their upper bounds.

1. If this is a warm start, $\mathcal{F}$, $\mathcal{L}$ and $\mathcal{U}$ were initialized externally; set each component of x whose index is in $\mathcal{F}$ to the average of its lower and upper bounds, and set each component of x whose index is in $\mathcal{L}$ and $\mathcal{U}$ to its corresponding bound. Otherwise, set $\mathcal{F} = \mathcal{U} = \emptyset$, $\mathcal{L} = \{1, \cdots, n\}$, and set every element of x to its lower bound.

2. Compute $w = A^T(b - Ax)$, the negative of the gradient of the squared objective.

3. If $\mathcal{F} = \{1, \cdots, n\}$, or, if $w_j \le 0$ for all $j \in \mathcal{L}$ and $w_j \ge 0$ for all $j \in \mathcal{U}$, go to step 12. (This is the Kuhn-Tucker test for convergence.)

4. Find $t^* = \arg\max_{t \in \mathcal{L} \cup \mathcal{U}} s_t w_t$, where $s_t = 1$ if $t \in \mathcal{L}$ and $s_t = -1$ if $t \in \mathcal{U}$. If t^* is not unique, break ties arbitrarily.

5. Move t^* to the set $\mathcal{F}$.

6. Let b' be the data vector less the predictions of the bound variables; *i.e.*, $b'_j = b_j - \sum_{k \in \mathcal{L} \cup \mathcal{U}} A_{jk} x_j$. Let A' be the matrix composed of those columns of A whose indices are in $\mathcal{F}$. Let j' denote the index of the column of A' corresponding to the original index $j \in \mathcal{F}$. Find $z = \arg\min \|A'z - b'\|_2^2$.

7. If $l_j < z_{j'} < u_j$ for all j', set $x_j = z_{j'}$ and go to step 2.

8. Let $\mathcal{J}$ be the set of indices of components of z that are out-of-bounds. Define

$$q' = \arg\min_{j' \in \mathcal{J}} \min \left\{ \left| \frac{l_j - x_j}{z_{j'} - x_j} \right|, \left| \frac{u_j - x_j}{z_{j'} - x_j} \right| \right\}.$$

9. Set

$$\alpha = \min \left\{ \left| \frac{l_q - x_q}{z_{q'} - x_q} \right|, \left| \frac{u_q - x_q}{z_{q'} - x_q} \right| \right\}$$

10. Set $x_j := x_j + \alpha(z_{j'} - x_j)$ for all $j \in \mathcal{F}$.

11. Move to $\mathcal{L}$ the index of every component of x at or below its lower bound. Move to $\mathcal{U}$ the index of every component at or above its upper bound. Go to step 6.

12. Done.

As noted by Lawson and Hanson, roundoff errors in computing w may cause a component on a bound to appear to want to become free, yet when the component is added to the free set, it moves away from the feasible region. When that occurs, the component is not freed, the corresponding component of w is set to zero, and the program returns to step 3 (the Kuhn-Tucker test).

When the solution of the unconstrained least-squares problem in step 6 is infeasible, it is used in a convex interpolation with the previous solution to obtain a feasible vector (as in NNLS) in step 11. The constant in this interpolation is computed to put at least one component of the new iterate x on a bound. However, because of roundoff, sometimes

no interpolated component ends up on a bound. Then in step 11, the component that determined the interpolation constant in step 8 is forced to move to the appropriate bound. This guarantees that what Lawson and Hanson call 'Loop B' is finite. Also following Lawson and Hanson, any component remaining infeasible at the end of step 11 is moved to its nearer bound.

There are several differences between NNLS and BVLS that improve numerical stability: Our implementation of BVLS uses the QR decomposition to solve the unconstrained least-squares problem in step 6, as does NNLS. While NNLS updates the QR decomposition each time step 6 is entered, for efficiency, we compute the decomposition from scratch each time step 6 is executed, for stability. We also assume the solution is essentially optimal if the norm of the residual vector is less than a small fraction of the norm of the original data vector b (10^{-12} in the code below); this test might be removed by the more cautious.

If the columns of A passed to the QR routine at step 6 are linearly dependent, the new component is not moved from its bound: the corresponding component of w is set to zero, and control returns to step 3, the Kuhn-Tucker test. When the columns of A are nearly linearly dependent, we have observed cycling of free components: a component just moved to a bound tries immediately to become free; the least-squares step (6) returns a feasible value and a different component is bound. This component immediately tries to become free again, and the original component is moved back to its previous bound. We have taken two steps to avoid this problem. First, the column of the matrix A corresponding to the new potentially free component is passed to QR as the last column of its matrix. This ordering tends to make a component recently moved to a bound fail the "round-off" test mentioned above. Second, we have incorporated a test that prohibits short cycles. If the most recent successful change to the free set was to bind a particular component, that component can not be the next to be freed. This test occurs just after the Kuhn-Tucker test (step 3).

An ANSI FORTRAN77 implementation of BVLS is in the appendix.

3 Using BVLS to Solve Minimum l_1 and l_∞ Problems

Consider the problem *bvmm(p)* (bounded-variable minimum misfit in the p norm):

$$\min_{l \le x \le u} \|Ax - b\|_p. \tag{6}$$

The problem *bvls* is the case $p = 2$. The cases $p = 1$ and $p = \infty$ can be written as linear programs, but can also be solved iteratively using algorithm BVLS. For reference, we give the linear programming formulations of these problems.

Problem *bvmm(1)* is solved by the following linear program:

$$\min \sum_{j=1}^{m} (s_j + t_j) \tag{7}$$

subject to the inequality constraints

$$l \le x \le u, \;\; 0 \le s, \;\; \text{and} \;\; 0 \le t, \tag{8}$$

where $x \in \mathbf{R}^n$, and $s, t \in \mathbf{R}^m$; and the linear equality constraints

$$Ax - b + s - t = 0. \tag{9}$$

The proof of the equivalence of this linear program to *bvmm(1)* is just the observation that $\|s-t\|_1 \le \sum_j (s_j + t_j)$, and that whatever $s_j - t_j$ may be, equality is possible without changing the fit to the data or violating the inequality constraints.

Problem *bvmm(∞)* is solved by the linear program:

$$\min r \tag{10}$$

subject to the linear inequality constraints

$$r \ge 0, \;\; l \le x \le u, \;\; \text{and} \;\; -\mathbf{1}r \le s \le \mathbf{1}r, \tag{11}$$

where $r \in \mathbf{R}$, $x \in \mathbf{R}^n$, $s \in \mathbf{R}^m$, and $\mathbf{1}$ is an m-vector of ones; and the linear equality constraints

$$Ax - b + s = 0. \tag{12}$$

3.1 Solving Problem *bvmm(1)* With BVLS

A computationally effective strategy is to mimic the linear program above, using large weights to impose the equality constraints. Let $\tilde{A}$ be the matrix A with its rows renormalized to unit Euclidean norm (to improve numerical stability), and let $\tilde{b}$ be the vector b scaled by the same constants as the rows of $\tilde{A}$. Define the matrix

$$G \equiv \begin{bmatrix} \tilde{A} & I & -I \\ \mathbf{0} & \gamma\mathbf{1} & \gamma\mathbf{1} \end{bmatrix}, \tag{13}$$

where I is an m by m identity matrix, $\mathbf{0}$ is a row vector of n zeros, $\mathbf{1}$ is a row-vector of m ones, and $\gamma << 1$ is a positive scalar discussed later. Define the $m+1$-vector

$$d \equiv \begin{bmatrix} \tilde{b} \\ 0 \end{bmatrix}, \tag{14}$$

and the $n + 2m$-vectors

$$e \equiv \begin{bmatrix} l \\ 0 \end{bmatrix}, \quad \text{and} \quad f \equiv \begin{bmatrix} u \\ \text{“}\infty\text{”} \end{bmatrix}. \tag{15}$$

Here “∞” is a very large number. Now consider the *bvls* problem

$$\min \|Gz - d\|_2 \text{ such that } e \leq z \leq f, \tag{16}$$

where $z \in \mathbf{R}^{n+2m}$. If γ is sufficiently small, the equations involving the matrix A will be satisfied with high accuracy, while the least-squares misfit will derive almost entirely from the last row, which is the l_1 misfit. A different approach is given below in section on 4.1.

3.2 Solving Problem *bvmm(∞)* With BVLS

The strategy we employ to solve *bvmm(∞)* as a sequence of *bvls* problems is to find the largest constant r such that constraining the infinity-norm misfit to be at most r still yields a feasible problem. In implementations, we have used a bisection search to find the smallest the smallest positive value of r such that the *bvls* problem

$$\min_{l \leq x \leq u,\ -r\mathbf{1} \leq s \leq r\mathbf{1}} \left\| \begin{bmatrix} A & I \end{bmatrix} \cdot \begin{bmatrix} x \\ s \end{bmatrix} - b \right\|_2^2 \approx 0, \tag{17}$$

(within machine precision) where **1** is an m-vector of ones. This strategy is quite stable numerically and less demanding on memory than introducing additional slack variables for each data relation. Numerical stability can be further improved by normalizing each row of A to unit Euclidean length, and adjusting the constraints on the vector s accordingly. The number of iterations in the search for r can be reduced substituting for the bisection an algorithm that uses derivatives. It is clear from set inclusions that the *bvls* misfit is a monotonic function of r, so the searches are straightforward.

In all of *bvmm(p)*, $p = 1, 2, \infty$, it is trivial to incorporate weights on the data to account for unequal error variances.

4 Solving Problem *blf(p)* With BVLS

The problems *blf(p)* with $p = 1, 2$, and ∞ can be solved by solving a sequence of related *bvls* problems. The following subsections describe strategies we have used successfully.

It is useful to have *a priori* bounds on $c \cdot x$, which we can find by ignoring the misfit constraints:

$$c^- \equiv \sum_{j:c_j \leq 0} c_j u_j + \sum_{j:c_j > 0} c_j l_j \leq c \cdot x \leq \sum_{j:c_j \geq 0} c_j u_j + \sum_{j:c_j < 0} c_j l_j \equiv c^+. \tag{18}$$

4.1 $p = 1$

Problem *blf(1)*, can be solved by standard linear programming techniques. However, we have found the following strategy to be faster and more reliable than naive simplex algorithms, such as that in [6].

One may test for feasibility, *i.e.* the existence of an x_0 satisfying $l \leq x_0 \leq u$ and $\|Ax_0 - b\|_1 \leq \chi$, by solving the following *bvls* problem:

Let

$$G \equiv \begin{bmatrix} A & I & -I & 0 \\ \mathbf{0} & \mathbf{1} & \mathbf{1} & 1 \end{bmatrix} \tag{19}$$

In this definition, the matrices I are m by n matrices with ones on the diagonal and zero elsewhere; **0** in the top row is a column vector of m zeros; **0** in the lower row is a row vector

of n zeros; the first two **1**'s in the lower row are row vectors of m zeros, and the last 1 is just the scalar 1.

Let d be the vector

$$d \equiv \begin{bmatrix} b \\ \chi \end{bmatrix} \tag{20}$$

and let e and f be given by

$$e \equiv \begin{bmatrix} l \\ \mathbf{0} \end{bmatrix}, \tag{21}$$

where $\mathbf{0}$ is a column vector of $2m+1$ zeros, and

$$f \equiv \begin{bmatrix} u \\ \text{“}\infty\text{”} \\ \chi \end{bmatrix}, \tag{22}$$

where “∞” is a $2m$-vector of numbers large enough that the bounds can not be active. It is possible to compute how large these numbers need to be *a priori* from A and b; alternatively, one can use trial values and verify *a posteriori* whether the bounds are active, and increase the value if they are.

The problem *blf(1)* is feasible if the *bvls* problem

$$\min_{y \in \mathbf{R}^{n+2m+1}} \|Gy - h\|_2 \text{ s.t. } e \leq y \leq f \tag{23}$$

has the value zero within machine precision. Then the first n elements of the solution vector y comprise a feasible point for the problem *blf(1)*, if one exists. This also gives an alternative algorithm for solving *bvmm(1)*: as in the algorithm for *bvmm(∞)*, use a bisection search to find the smallest value of χ for which the *bvls* problem has value 0 within machine precision.

The information in the `istate` vector can be used to warm-start the iterations about to be described. Re-define

$$G \equiv \begin{bmatrix} A & I & -I & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{1} & 1 & 0 & 0 \\ c & \mathbf{0} & \mathbf{0} & 0 & 1 & -1 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & 0 & \gamma & \gamma \end{bmatrix} \tag{24}$$

In this definition, the matrices I are as before, and the vectors 0 in the top row are column-vectors of m zeros. In the second row, the first 0 is a row-vector of n zeros; the first two

vectors **1** are row-vectors of m ones; and the third 1 and the last two 0's are scalars. In the third row, c is the objective n-vector (written as a row-vector); the first two **0**'s are row vectors of m zeros; and the third 0, the 1 and the -1 are scalars. In the last row, the first **0** is a row-vector of n zeros; the next two **0**'s are row-vectors of m zeros; and the last three entries are the scalars 0, γ and γ. The scalar γ is a small positive number used to downweight the last row relative to those above, which we want to treat as equality constraints. The dimension of G is thus $m+3$ by $n+2m+3$.

Let d now be the $m+3$-vector

$$d \equiv \begin{bmatrix} b \\ \chi \\ c^{\pm} \\ 0 \end{bmatrix}, \tag{25}$$

where c^+ is used to maximize $c \cdot x$, and c^- is used to minimize $c \cdot x$ (see the definitions 18). Let e and f be given by

$$e \equiv \begin{bmatrix} l \\ 0 \end{bmatrix}, \tag{26}$$

where 0 is a column vector of $2m+3$ zeros, and

$$f \equiv \begin{bmatrix} u \\ \text{“}\infty\text{”} \\ \chi \\ \infty \\ \infty \end{bmatrix}, \tag{27}$$

where “∞” is a $2m$-vector of numbers large enough that the constraints will never be active, and the last two entries ∞ are two more such scalars.

Let x^* denote the n-vector consisting of the first n elements of the solution of the *bvls* problem

$$\arg\min \{ \|Gz - d\| : e \le z \le f \}. \tag{28}$$

If γ is chosen well, $c \cdot x^*$ gives the optimal value of *blf(1)*. One should verify *a posteriori* that the l_1 misfit $\|Ax^* - b\|_1$ is adequately close to χ; if not, γ needs to be adjusted. A proof that this scheme works follows that given in the next section for *blf(2)*.

We remark that the dimension of the matrix G is larger than it needs to be; it is possible to eliminate the last row and last two columns, substituting the row $[\gamma c\ 0\ 0\ 0]$ for $[c\ 0\ 0\ 0\ 1\ 1]$ and making corresponding changes to d, e, and f. However, we have found the scheme spelt out above to be more stable and less sensitive to the choice of γ.

4.2 $p = 2$

This problem can be solved relatively easily using BVLS and a scheme that lies conceptually between the Lagrangian dual and the use of linear constraints directly. (See Rust and Burris [8] for a different algorithm.) The basic idea is to minimize the misfit to the data subject to the inequality constraints on x and the constraint $c \cdot x = \gamma$. The value of γ is varied from the "best-fitting" value to determine the range of values for which the minimum misfit is at most χ.

We can find the "best-fitting" value of γ by solving the *bvls* problem

$$x_0 = \arg \min_{l \le x \le u} \|Ax - b\|_2^2, \tag{29}$$

and setting $\gamma_0 \equiv c \cdot x_0$.

Consider finding $\min c \cdot x$; the case for the maximum follows by changing the sign of c or other obvious modifications. Define the matrix

$$G \equiv \begin{bmatrix} A \\ \alpha c \end{bmatrix} \tag{30}$$

and the vector

$$d \equiv \begin{bmatrix} b \\ \alpha\gamma \end{bmatrix}. \tag{31}$$

Now consider

$$\Phi(\gamma) \equiv \min_{l \le x \le u} \|Gx - d\|_2^2, \tag{32}$$

and let $x^*(\gamma)$ denote the minimizer. This is a *bvls* problem. We assert that for $\gamma_t < \gamma_0$, $c \cdot x^*(\gamma)$ is the solution to

$$\min c \cdot x \quad \text{such that } l \le x \le u \text{ and } \|Ax - b\|_2 \le \|Ax^*(\gamma) - b\|_2. \tag{33}$$

The proof just relies on the convexity of the linear functional $c \cdot x$, the strict convexity of $\|Ax - b\|_2^2$, and the convexity of the set $l \leq x \leq u$. One may thus solve *blf(2)* by finding the smallest and largest values of γ such that $\|Ax^*(\gamma) - b\|_2 \leq \chi$. The search is straightforward since $\|Ax^*(\gamma) - b\|$ increases monotonically with $|\gamma - \gamma_0|$ (the functional is quasiconvex in γ). If, during the search, $c \cdot x^*(\gamma) \leq c^-$ (or $c \cdot x^*(\gamma) \geq c^+$) while $\|Ax^*(\gamma) - b\| < \chi$, then the minimum (maximum, respectively) value of $c \cdot x$ is c^- (c^+). The warm-start feature of BVLS affords large economies in solving the sequence of *bvls* problems for different values of γ, especially since the sets of components at their upper and lower bounds change gradually as γ varies.

The positive constant α is chosen to enhance numerical stability; we have found that for n up to about 200, $\alpha \approx 10^{-3}$ works well in double precision on 32 bit machines. One should allow some tolerance in attaining the exact value of χ (*e.g.,* 1% of the nominal value) to avoid excessive searching. This may be justified by remembering that in practice one picks χ using (uncertain) estimates of the standard deviations of the data errors.

4.3 $p = \infty$

This problem can also be solved directly using linear programming; we have found the following approach using BVLS to be faster and more stable than simple simplex methods. Reducing problem *blf*(∞) to a *bvls* problems is similar to the treatment for $p = 1$.

As noted in section 3.2, to find a feasible point one may solve

$$\arg \min_{e \leq z \leq f} \|Gz - d\|_2^2, \tag{34}$$

where

$$G \equiv \begin{bmatrix} \Lambda A & I & \mathbf{1} \end{bmatrix}, \tag{35}$$

(Λ is a diagonal matrix introduced to improve numerical stability; its entries are the reciprocals of the Euclidean norms of the corresponding rows of A; I is an m by m identity matrix, and $\mathbf{1}$ is a column-vector of n ones)

$$e \equiv \begin{bmatrix} l \\ -\Lambda \cdot \chi \end{bmatrix}, \tag{36}$$

where χ is an m-vector all of whose entries are χ, and

$$f \equiv \begin{bmatrix} u \\ \Lambda\chi \end{bmatrix}. \tag{37}$$

The first n elements of z give a feasible point for the infinity-norm problem.

Let z^* solve the *bvls* problem

$$\min \|Gz - d\|_2^2 \tag{38}$$

such that $e \leq z \leq f$, where

$$G \equiv \begin{bmatrix} \Lambda A & I & 0 & 0 \\ c & \mathbf{0} & 1 & -1 \\ \mathbf{0} & \mathbf{0} & \gamma & \gamma \end{bmatrix}, \tag{39}$$

$$e \equiv \begin{bmatrix} l \\ -\Lambda \cdot \chi \\ 0 \\ 0 \end{bmatrix}, \tag{40}$$

and

$$f \equiv \begin{bmatrix} u \\ \Lambda \cdot \chi \\ \infty \\ \infty \end{bmatrix}. \tag{41}$$

Let x^* denote the n-vector composed of the first n elements of z^*. If α is chosen well, $c \cdot z^*$ solves *blf(∞)*; again, one should check *a posteriori* to verify that $\|Ax^* - b\|_\infty$ is adequately close to χ.

5 Application to Monotone Regression and Calibration

The monotone regression problem is as follows: we observe

$$d_j = f(t_j) + e_j, \quad j = 1, \cdots, n \tag{42}$$

where f is monotone (increasing) function of t, $\{t_j\}_{j=1}^n$ are known, and $\{e_j\}$ are independent random errors with mean zero and known variances $\{\sigma_j^2\}$. Without loss of generality, we assume that $t_j \leq t_{j+1}$, $j = 1, \cdots, n-1$. We assume for simplicity of exposition that the t_j are all distinct, and that the errors e_j are independent, identically distributed (iid) Gaussian variables with common variance 1. See [7] for more on monotone regression.

From these data, we would like to find a confidence region for f; *i.e.* a pair of functions $f^-(t)$ and $f^+(t)$ so that $f(t) \in [f^-(t), f^+(t)]$, simultaneously $\forall t \in \mathbf{R}$, with $1-\alpha$ confidence. Such an envelope can be found using the "strict bounds" approach [9, 1] by solving certain infinite-dimensional optimization problems.

5.1 Theory

The set

$$\Xi \equiv \left\{\xi \in \mathbf{R}^n : \|\xi - d\|_2^2 \leq \chi^2_{\alpha,n}\right\} \tag{43}$$

is a $1-\alpha$ confidence region for the vector $\vec{f} \equiv (f(t_1), \cdots, f(t_n))^T$ of "noise-free" observations, if $\chi^2_{\alpha,n}$ is the α percentage point of the chi-square distribution with n degrees of freedom. The preimage of that set, the set of all functions $g : \mathbf{R} \to \mathbf{R}$ defined by

$$\mathcal{D} \equiv \{g : (g(t_1), \cdots, g(t_n))^T \in \Xi\} \tag{44}$$

is therefore a $1-\alpha$ confidence region for the unknown regression function f. Let $\mathcal{C}$ be the set of monotone functions; we know *a priori* that $f \in \mathcal{C}$. Therefore $\mathcal{C} \cap \mathcal{D}$ is a $1-\alpha$ confidence region for f.

Now consider the optimization problems

$$f^-(t) \equiv \arg \inf_{g \in \mathcal{C} \cap \mathcal{D}} g(t), \tag{45}$$

and

$$f^+(t) \equiv \arg \sup_{g \in \mathcal{C} \cap \mathcal{D}} g(t). \tag{46}$$

This pair of functions gives the desired confidence envelope. The optimization problems can be reduced to finite-dimensional problems of type *blf(2)* (see below), which makes them tractable.

If $g \in \mathcal{C} \cap \mathcal{D}$, then so are the functions

$$g^-(t) \equiv \begin{cases} g(t_1), & t \le t_1 \\ g(t_j), & t_{j-1} < t \le t_j, \ \ j = 2, \cdots, n \\ -\infty, & t > t_n, \end{cases} \tag{47}$$

and

$$g^+(t) \equiv \begin{cases} \infty, & t < t_1 \\ g(t_j), & t_j \le t < t_{j+1}, \ \ j = 1, \cdots, n-1 \\ g(t_n), & t > t_n. \end{cases} \tag{48}$$

Furthermore, $g^-(t) \le g(t) \le g^+(t), \forall t \in \mathbf{R}$, whatever be $g \in \mathcal{C} \cap \mathcal{D}$. It follows that it suffices to take the infimum and supremum over the subspaces spaces of piecewise constant functions with discontinuities at $\{t_j\}$, and that $f^-(t)$ and $f^+(t)$ are piecewise constant as well.

We may thus find $f^-(t_j)$ by solving the finite-dimensional problems

$$f^-(t_j) \equiv \min\{\gamma_j : \|\gamma - d\|_2^2 \le \chi^2_{\alpha,n} \ \text{ and } \ \gamma_k \ge \gamma_{k+1}, \ k = 1, \cdots, n-1\}. \tag{49}$$

Similarly,

$$f^+(t_j) \equiv \max\{\gamma_j : \|\gamma - d\|_2^2 \le \chi^2_{\alpha,n} \ \text{ and } \ \gamma_k \ge \gamma_{k+1}, \ k = 1, \cdots, n-1\}. \tag{50}$$

The bounds at these n points for $j = 1, \cdots, n$ can be interpolated and extrapolated using (47) and (48) and still maintain simultaneous $1 - \alpha$ coverage probability $\forall t \in \mathbf{R}$.

5.2 Reduction to *blf(2)*

These finite-dimensional optimization problems can be written as problems of type *blf(2)* by making the following identifications. Let x be the vector

$$x \equiv (\gamma_1, \gamma_2 - \gamma_1, \cdots, \gamma_n - \gamma_{n-1})^T. \tag{51}$$

Then $f^-(t)$ is given by a sum of the form

$$f^-(t) = \sum\{x_j : t_j \le t\}, \tag{52}$$

while f^+ is given by a sum of the form

$$f^+(t) = \sum\{x_j : t_j < t\}. \tag{53}$$

The monotonicity of f^- and f^+ is ensured by the linear inequality constraints $0 \geq x_j$, $j = 2, \cdots, n$ for f is monotone decreasing (for monotone increasing f, make the obvious changes).

Thus (for monotone decreasing f) if we define A to be the n by n matrix:

$$A \equiv \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 0 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 1 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 1 & 1 & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & 1 & 1 & 1 & 1 & \cdots & 1 \end{bmatrix}, \tag{54}$$

define l and u by

$$l_j = -\infty, \quad j = 1, \cdots, n, \tag{55}$$

and

$$u_1 = \infty; \quad u_j = 0, \quad j = 2, \cdots, n; \tag{56}$$

and define $c^-(t)$ and $c^+(t)$ by

$$c_j^-(t) = \begin{cases} 1, & t_j \leq t \\ 0, & \text{otherwise,} \end{cases} \tag{57}$$

and

$$c_j^+(t) = \begin{cases} 1, & t_j < t \\ 0, & \text{otherwise.} \end{cases} \tag{58}$$

With these definitions, the optimization problems become the *blf(2)* problems:

$$\begin{matrix} \min \\ \max \end{matrix} \; c^{\pm}(t) \cdot x \quad \text{such that} \quad l \leq x \leq u \quad \text{and} \quad \|Ax - d\|_2 \leq \chi. \tag{59}$$

In practice, one replaces $-\infty$ in l by a large negative number, and ∞ in u by a large positive number.

5.3 Computations for Monotone Regression

We implemented FORTRAN 77 code for *blf(2)* using a bisection search to find the "target" value of $c^{\pm}(t) \cdot x$ that yielded a minimum two-norm misfit $0.99\chi^2 \leq \|Az - d\|_2^2 \leq 1.01\chi^2$. We

applied the algorithm to two sets of artificial data. In both, the data errors were iid pseudo-random normal variates with mean zero and variance 1, and the data were equispaced (this is irrelevant since the extremal functions and bounds are piecewise constant). The first set took $t_j = \pi\frac{j-1}{200}$, $j = 1, \cdots, 100$ and $f(t) \equiv 100 \cos t$; the second set took $t_j = \frac{j-1}{100}$, $j = 1, \cdots, 100$, and $f(t) \equiv 50$. The target value for χ^2 was $\chi^2_{.05,100} = 124.1$. The values "$\pm\infty$" in l and u were replaced by $\pm 10,000$. The data and the resulting 95% confidence regions are plotted in Figures 1 and 2. Solving the 200 optimization problems to find upper and lower bounds at each t_j took approximately 4 cpu minutes on a SparcStation 2 for each data set.

The bounds are narrow enough to be interesting, although the signal-to-noise ratio (roughly 50:1) may be optimistic for many problems. Note that the bounds are much narrower in the second case than in the first, since the constant regression function comes closer to violating the monotonocity constraint than the cosine does. Also, note that due to the nonlinearity of the procedure, while the bounds for the constant regression function are nearly symmetric, they are not exactly so, since the data errors vary.

5.4 Calibration

This approach to monotone regression can be "inverted" to give confidence intervals for calibration problems. In a calibration problem, we observe m ordered pairs (t_j, y_j), $j = 1, \cdots, m$, and the value y_0. We assume $y_j = f(t_j) + e_j$, where f is monotonic decreasing and $\{e_j\}_{j=0}^m$ are iid $N(0,1)$. We seek a $1-\alpha$ confidence interval for t_0.

We solve this problem by first finding a confidence interval for $f(t_0)$. Let α' and α'' be any pair of numbers satisfying $1-\alpha = (1-\alpha')(1-\alpha'')$, and let $[\phi^-, \phi^+]$ be a $1-\alpha'$ confidence interval for $f(t_0)$. Let $f^-(t)$ and $f^+(t)$ be the lower and upper boundaries of a $1-\alpha''$ simultaneous confidence envelope for $f(t)$ using the data y_j, $j = 1, \cdots, m$, computed as described above for monotone regression. Now define

$$t^+ \equiv \sup\{t : f^-(t) \geq \phi^-\} \tag{60}$$

and

$$t^- \equiv \inf\{t : f^+(t) \leq \phi^+\}. \tag{61}$$

Since e_0 is independent of the other errors, $[t^-, t^+]$ is a $1-\alpha$ confidence interval for t_0.

A Fortran Code for BVLS

This routine calls QR, a FORTRAN subroutine that solves unconstrained least-squares problems using the QR decomposition via Householder rotations. Code for QR is given in the next appendix.

```
c=====================================================================
      subroutine bvls(key, m, n, a, b, bl, bu, x, w, act, zz, istate,
     +  loopA)
c=====================================================================
      implicit double precision (a-h, o-z)
c
c**** calls qr
c--------------------Bounded Variable Least Squares--------------------
c
c        Robert L. Parker and Philip B. Stark    Version 3/19/90
c
c  Solves the problem:
c
c          min  || a.x - b ||     such that   bl <= x <= bu
c                            2
c    where
c               x  is an unknown n-vector
c               a  is a given m by n matrix
c               b  is a given  m-vector
c               bl is a given n-vector of lower bounds on the
c                                components of x.
c               bu is a given n-vector of upper bounds on the
c                                components of x.
c
c
c---------------------------------------------------------------------
c    Input parameters:
```

```
c
c  m, n, a, b, bl, bu   see above.   Let mm=min(m,n).
c
c  If key = 0, the subroutine solves the problem from scratch.
c
c  If key > 0 the routine initializes using the user's guess about
c   which components of  x  are 'free', i.e. are stricly within their
c   bounds, which are at their lower bounds, and which are at their
c   upper bounds.  This information is supplied through the array
c   istate.  istate(n+1) should contain the total number of components
c   at their bounds (the 'bound variables').  The absolute values of the
c   first nbound=istate(n+1) entries of  istate  are the indices
c   of these 'bound' components of  x.  The sign of istate(j), j=1,...,
c   nbound, indicates whether  x(|istate(j)|) is at its upper or lower
c   bound.  istate(j) is positive if the component is at its upper
c   bound, negative if the component is at its lower bound.
c   istate(j), j=nbound+1,...,n  contain the indices of the components
c   of  x  that are free (i.e. are expected to lie strictly within
c   their bounds).  When key > 0, the routine initially sets the free
c   components to the averages of their upper and lower bounds:
c   x(j)=(bl(j)+bu(j))/2, for j in the free set.
c
c-----------------------------------------------------------------------
c    Output parameters:
c
c  x       the solution vector.
c
c  w(1)    the minimum 2-norm || a.x-b ||.
c
c  istate  vector indicating which components of  x  are free and
c          which are at their bounds (see the previous paragraph).
c          istate can be supplied to the routine to give it a good
```

```
c          starting guess for the solution.
c
c  loopA   number of iterations taken in the main loop, Loop A.
c
c-----------------------------------------------------------------------
c    Working  arrays:
c
c  w      dimension n.              act      dimension m*(mm+2).
c  zz     dimension m.              istate   dimension n+1.
c
c-----------------------------------------------------------------------
c  Method: active variable method along the general plan of NNLS by
c  Lawson & Hanson, "Solving Least Squares Problems," Prentice-Hall, 1974.
c  See Algorithm 23.10.
c  Step numbers in comment statements refer to Lawson and Hanson's scheme.
c
c-----------------------------------------------------------------------
c
c
      dimension a(m,n), b(m), x(n), bl(n), bu(n)
c     dimension w(n), act(m,min(m,n)+2), zz(m), istate(n+1)
      dimension w(n), act(m,m+2), zz(m), istate(n+1)
c
      data eps/1.0d-12/
c
c----------------------First Executable Statement----------------------
c
c  Step 1.  Initialize everything--free and bound sets, initial
c   values, etc.
c
c  Initialize flags, etc.
      mm=min(m,n)
```

```
      mm1 = mm + 1
      jj = 0
      ifrom5 = 0
c  Check consistency of given bounds  bl, bu.
      bdiff = 0.0
      do 1005 j=1, n
        bdiff=max(bdiff, bu(j)-bl(j))
        if (bl(j) .gt. bu(j)) then
          print *,' Inconsistent bounds in BVLS. '
          stop
        endif
 1005 continue
      if (bdiff .eq. 0.0) then
        print *,' No free variables in BVLS--check input bounds.'
        stop
      endif
c
c  In a fresh initialization (key = 0) bind all variables at their lower
c   bounds.  If (key != 0), use the supplied  istate  vector to
c   initialize the variables.  istate(n+1) contains the number of
c   bound variables.  The absolute values of the first
c   nbound=istate(n+1) entries of  istate  are the indices of the bound
c   variables.  The sign of each entry determines whether the indicated
c   variable is at its upper (positive) or lower (negative) bound.
      if (key .eq. 0) then
        nbound=n
        nact=0
        do 1010 j=1, nbound
          istate(j)=-j
 1010   continue
      else
        nbound=istate(n+1)
```

```
      endif
      nact=n - nbound
      if ( nact .gt. mm ) then
        print *, ' Too many free variables in BVLS starting solution!'
        stop
      endif
      do 1100 k=1, nbound
        j=abs(istate(k))
        if (istate(k) .lt. 0) x(j)=bl(j)
        if (istate(k) .gt. 0) x(j)=bu(j)
 1100 continue
c
c  In a warm start (key != 0) initialize the free variables to
c   (bl+bu)/2.  This is needed in case the initial qr results in
c   free variables out-of-bounds and Steps 8-11 get executed the
c   first time through.
      do 1150 k=nbound+1,n
        kk=istate(k)
        x(kk)=(bu(kk)+bl(kk))/2
 1150 continue
c
c  Compute bnorm, the norm of the data vector b, for reference.
      bsq=0.0
      do 1200 i=1, m
        bsq=bsq + b(i)**2
 1200 continue
      bnorm=sqrt(bsq)
c
c-----------------------------Main Loop--------------------------------
c
c  Initialization complete.  Begin major loop (Loop A).
      do 15000 loopA=1, 3*n
```

```
c
c  Step 2.
c  Initialize the negative gradient vector w(*).
 2000 obj=0.0
      do 2050 j=1, n
        w(j)=0.0
 2050 continue
c
c  Compute the residual vector b-a.x , the negative gradient vector
c   w(*), and the current objective value obj = || a.x - b ||.
c   The residual vector is stored in the mm+1'st column of act(*,*).
      do 2300 i=1, m
        ri=b(i)
        do 2100 j=1, n
          ri=ri - a(i,j)*x(j)
 2100   continue
        obj=obj + ri**2
        do 2200 j=1, n
          w(j)=w(j) + a(i,j)*ri
 2200   continue
        act(i,mm1)=ri
 2300 continue
c
c  Converged?  Stop if the misfit << || b ||, or if all components are
c   free (unless this is the first iteration from a warm start).
      if (sqrt(obj) .le. bnorm*eps .or.
     &  (loopA .gt. 1 .and. nbound .eq. 0)) then
         istate(n+1)=nbound
         w(1)=sqrt(obj)
         return
      endif
c
```

```
c  Add the contribution of the free components back into the residual.
      do 2500 k=nbound+1, n
        j=istate(k)
        do 2400 i=1, m
          act(i,mm1)=act(i,mm1) + a(i,j)*x(j)
 2400   continue
 2500 continue
c
c  The first iteration in a warm start requires immediate qr.
      if (loopA .eq. 1 .and. key .ne. 0) goto 6000
c
c  Steps 3, 4.
c  Find the bound element that most wants to be free.
 3000 worst=0.0
      it=1
      do 3100 j=1, nbound
         ks=abs(istate(j))
         bad=w(ks)*sign(1, istate(j))
         if (bad .lt. worst) then
            it=j
            worst=bad
            iact=ks
         endif
 3100 continue
c
c  Test whether the Kuhn-Tucker condition is met.
      if (worst .ge. 0.0 ) then
         istate(n+1)=nbound
         w(1)=sqrt(obj)
         return
      endif
c
```

```
c  The component  x(iact) most wants to become free.
c   If the last successful change in the free set was to move x(iact)
c   to a bound, don't free x(iact) now: set the derivative of the
c   misfit with respect to x(iact) to zero and return to the Kuhn-Tucker
c   test.
      if ( iact .eq. jj ) then
        w(jj)=0.0
        goto 3000
      endif
c
c  Step 5.
c  Undo the effect of the new (potentially) free variable on the
c   residual vector.
      if (istate(it) .gt. 0) bound=bu(iact)
      if (istate(it) .lt. 0) bound=bl(iact)
      do 5100 i=1, m
        act(i,mm1)=act(i,mm1) + bound*a(i,iact)
 5100 continue
c
c  Set flag ifrom5, indicating that Step 6 was entered from Step 5.
c   This forms the basis of a test for instability: the gradient
c   calculation shows that x(iact) wants to join the free set; if
c   qr puts x(iact) beyond the bound from which it came, the gradient
c   calculation was in error and the variable should not have been
c   introduced.
      ifrom5=istate(it)
c
c  Swap the indices (in istate ) of the new free variable and the
c   rightmost bound variable; 'free' that location by decrementing
c   nbound.
      istate(it)=istate(nbound)
      nbound=nbound - 1
```

```
      nact=nact + 1
      istate(nbound+1)=iact
c
      if (mm .lt. nact) then
        print *,' Too many free variables in BVLS!'
        stop
      endif
c
c  Step 6.
c  Load array  act  with the appropriate columns of  a  for qr.  For
c   added stability, reverse the column ordering so that the most
c   recent addition to the free set is in the last column.  Also
c   copy the residual vector from act(., mm1) into act(., mm1+1).
 6000 do 6200 i=1, m
        act(i,mm1+1)=act(i,mm1)
        do 6100 k=nbound+1, n
          j=istate(k)
          act(i,nact+1-k+nbound)=a(i,j)
 6100   continue
 6200 continue
c
      call qr(m, nact, act, act(1,mm1+1), zz, resq)
c
c  Test for linear dependence in qr, and for an instability that moves
c   the variable just introduced away from the feasible region
c   (rather than into the region or all the way through it).
c   In either case, remove the latest vector introduced from the
c   free set and adjust the residual vector accordingly.
c   Set the gradient component (w(iact)) to zero and return to
c   the Kuhn-Tucker test.
      if (resq .lt. 0.0
     &   .or. (ifrom5 .gt. 0 .and. zz(nact) .gt. bu(iact))
```

```
     &    .or. (ifrom5 .lt. 0 .and. zz(nact) .lt. bl(iact))) then
          nbound=nbound + 1
          istate(nbound)=istate(nbound)*sign(1.0d0, x(iact)-bu(iact))
          nact=nact - 1
          do 6500 i=1, m
            act(i,mm1)=act(i,mm1) - x(iact)*a(i,iact)
 6500     continue
          ifrom5=0
          w(iact)=0.0
          goto 3000
        endif
c
c  If Step 6 was entered from Step 5 and we are here, a new variable
c   has been successfully introduced into the free set; the last
c   variable that was fixed at a bound is again permitted to become
c   free.
        if ( ifrom5 .ne. 0 ) jj=0
        ifrom5=0
c
c   Step 7.  Check for strict feasibility of the new qr solution.
        do 7100 k=1, nact
          k1=k
          j=istate(k+nbound)
          if (zz(nact+1-k).lt.bl(j) .or. zz(nact+1-k).gt.bu(j)) goto 8000
 7100   continue
        do 7200 k=1, nact
          j=istate(k+nbound)
          x(j)=zz(nact+1-k)
 7200   continue
c  New iterate is feasible; go back to the top.
        goto 15000
c
```

```
c  Steps 8, 9.
 8000 alpha=2.0
      alf=alpha
      do 8200 k=k1, nact
        j=istate(k+nbound)
        if (zz(nact+1-k) .gt. bu(j))
     &    alf=(bu(j)-x(j))/(zz(nact+1-k)-x(j))
        if (zz(nact+1-k) .lt. bl(j))
     &    alf=(bl(j)-x(j))/(zz(nact+1-k)-x(j))
        if (alf .lt. alpha) then
          alpha=alf
          jj=j
          sj=sign(1.0, zz(nact+1-k)-bl(j))
        endif
 8200 continue
c
c  Step 10
      do 10000 k=1, nact
        j=istate(k+nbound)
        x(j)=x(j) + alpha*(zz(nact+1-k)-x(j))
10000 continue
c
c  Step 11.
c  Move the variable that determined alpha to the appropriate bound.
c   (jj is its index; sj is + if zz(jj)> bu(jj), - if zz(jj)<bl(jj) ).
c   If any other component of  x  is infeasible at this stage, it must
c   be due to roundoff.  Bind every infeasible component and every
c   component at a bound to the appropriate bound.  Correct the
c   residual vector for any variables moved to bounds.  Since at least
c   one variable is removed from the free set in this step, Loop B
c   (Steps 6-11) terminates after at most  nact  steps.
      noldb=nbound
```

```
      do 11200 k=1, nact
        j=istate(k+noldb)
        if (((bu(j)-x(j)) .le. 0.0) .or.
     &    (j .eq. jj .and. sj .gt. 0.0)) then
c  Move x(j) to its upper bound.
          x(j)=bu(j)
          istate(k+noldb)=istate(nbound+1)
          istate(nbound+1)=j
          nbound=nbound+1
          do 11100 i=1, m
             act(i,mm1)=act(i,mm1) - bu(j)*a(i,j)
11100     continue
        else if (((x(j)-bl(j)) .le. 0.0) .or.
     &    (j .eq. jj .and. sj .lt. 0.0)) then
c  Move x(j) to its lower bound.
          x(j)=bl(j)
          istate(k+noldb)=istate(nbound+1)
          istate(nbound+1)=-j
          nbound=nbound+1
          do 11150 i=1, m
             act(i,mm1)=act(i,mm1) - bl(j)*a(i,j)
11150     continue
        endif
11200 continue
      nact=n - nbound
c
c  If there are still free variables left repeat the qr; if not,
c    go back to the top.
      if (nact .gt. 0 ) goto 6000
c
15000 continue
c
```

```
      print *,' BVLS fails to converge! '
      stop
      end                                                              bvls
```

B FORTRAN Code for Subroutine QR

The subroutine QR solves unconstrained least-squares problems using the QR decomposition, found via Householder rotations. Details may be found in Lawson and Hanson [2].

```
c=====================================================================
      subroutine qr(m, n, a, b, x, resq)
      implicit double precision (a-h, o-z)                             *doub
c**** calls no other routines
c  Relies on FORTRAN77 do-loop conventions!
c  Solves over-determined least-squares problem  ax ~ b
c  where  a  is an  m by n  matrix,  b  is an m-vector .
c  resq  is the sum of squared residuals of optimal solution.  Also used
c  to signal error conditions - if -2 , system is underdetermined,  if
c  -1,  system is singular.
c  Method - successive Householder rotations.  See Lawson & Hanson -
c  Solving Least Squares Problems (1974).
c  Routine will also work when m=n.
c*****   CAUTION -  a and b  are overwritten by this routine.
      dimension a(m,n),b(m),x(n)
      double precision sum, dot
c
      resq=-2.0
      if (m .lt. n) return
      resq=-1.0
c   Loop ending on 1800 rotates  a  into upper triangular form.
      do 1800 j=1, n
c   Find constants for rotation and diagonal entry.
```

```
        sq=0.0
        do 1100 i=j, m
          sq=a(i,j)**2 + sq
 1100   continue
        if (sq .eq. 0.0) return
        qv1=-sign(sqrt(sq), a(j,j))
        u1=a(j,j) - qv1
        a(j,j)=qv1
        j1=j + 1
c  Rotate remaining columns of sub-matrix.
        do 1400 jj=j1, n
          dot=u1*a(j,jj)
          do 1200 i=j1, m
            dot=a(i,jj)*a(i,j) + dot
 1200     continue
          const=dot/abs(qv1*u1)
          do 1300 i=j1, m
            a(i,jj)=a(i,jj) - const*a(i,j)
 1300     continue
          a(j,jj)=a(j,jj) - const*u1
 1400   continue
c  Rotate  b  vector.
        dot=u1*b(j)
        do 1600 i=j1, m
          dot=b(i)*a(i,j) + dot
 1600   continue
        const=dot/abs(qv1*u1)
        b(j)=b(j) - const*u1
        do 1700 i=j1, m
          b(i)=b(i) - const*a(i,j)
 1700   continue
 1800 continue
```

```
c  Solve triangular system by back-substitution.
      do 2200 ii=1, n
        i=n-ii+1
        sum=b(i)
        do 2100 j=i+1, n
          sum=sum - a(i,j)*x(j)
 2100   continue
        if (a(i,i).eq. 0.0) return
         x(i)=sum/a(i,i)
 2200 continue
c  Find residual in overdetermined case.
      resq=0.0
      do 2300 i=n+1, m
        resq=b(i)**2 + resq
 2300 continue
      return
      end                                                              qr
c=====================================================================
```

References

[1] N.W. Hengartner and P.B. Stark. Conservative finite-sample confidence envelopes for monotone and unimodal densities. Technical Report 341, Dept. Stat. Univ. California, Berkeley, September 1992.

[2] C.W. Lawson and R.J. Hanson. *Solving Least Squares Problems.* John Wiley and Sons, Inc., New York, 1974.

[3] R.L. Parker. A theory of ideal bodies for seamount magnetism. *jgr*, 96:16101–16112, 1991.

[4] R.L. Parker and K. Whaler. Numerical methods for establishing solutions to the inverse problem of electromagnetic induction. *jgr*, 86:9574–9584, 1981.

[5] R.L. Parker and M.A. Zumberge. An analysis of geophysical experiments to test Newton's law of gravity. *Nature*, 342:39–31, 1989.

[6] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling. *Numerical Recipes in C.* Cambridge University Press, New York, 1988.

[7] T. Robertson, F.T. Wright, and R.L. Dykstra. *Order Restricted Statistical Inference.* John Wiley and Sons, Chichester, 1988.

[8] B.W. Rust and W.R. Burrus. *Mathematical Programming and the Numerical Solution of Linear Equations.* American Elsevier Pub. Co., Inc., New York, 1972.

[9] P.B. Stark. Inference in infinite-dimensional inverse problems: Discretization and duality. *J. Geophys. Res.*, 97:14,055–14,082, 1992.

[10] P.B. Stark and R.L. Parker. Smooth profiles from $\tau(p)$ and $X(p)$ data. *Geophys. J. R. Astron. Soc.*, 89:997–1010, 1987.

[11] P.B. Stark and R.L. Parker. Velocity bounds from statistical estimates of $\tau(p)$ and $X(p)$. *J. Geophys. Res.*, 92:2713–2719, 1987.

[12] P.B. Stark, R.L. Parker, G. Masters, and J.A. Orcutt. Strict bounds on seismic velocity in the spherical Earth. *J. Geophys. Res.*, 91:13,892–13,902, 1986.

Figure Captions.

Fig. 1: Simultaneous 95% confidence bounds in a monotone regression problem. The plusses are the original (synthetic) data, derived from a quarter of a cosine cycle, times 100, plus iid Gaussian noise with mean zero and variance 1. The bounds were found using BVLS as described in section 5.3.

Fig. 2: Same as Fig. 1, but the underlying regression function was the constant 50. Note that the bounds are narrower than in Fig. 1, since a constant function is closer to violating the monotonocity constraint than the quarter cosine wave in the previous example.

Figure 1

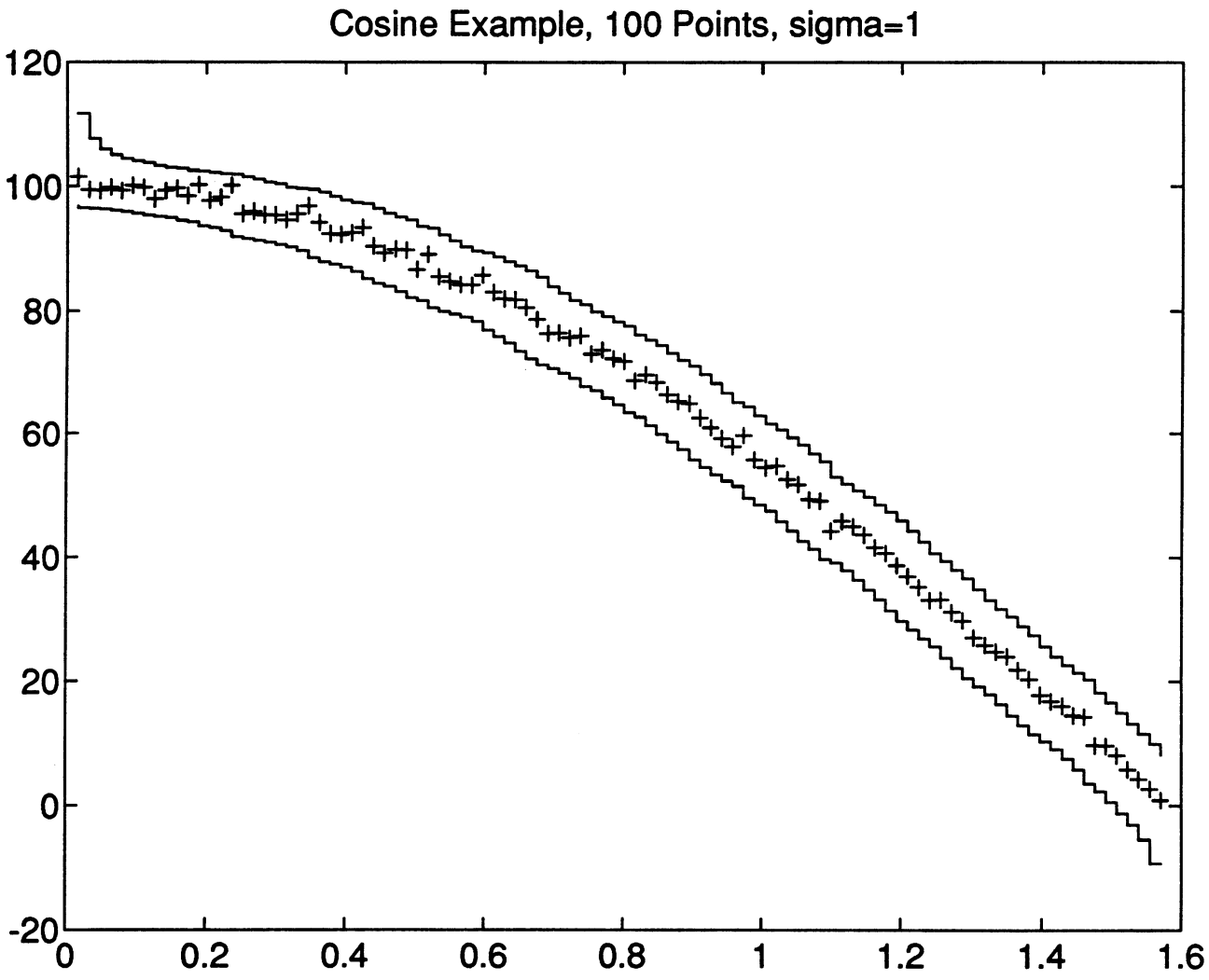
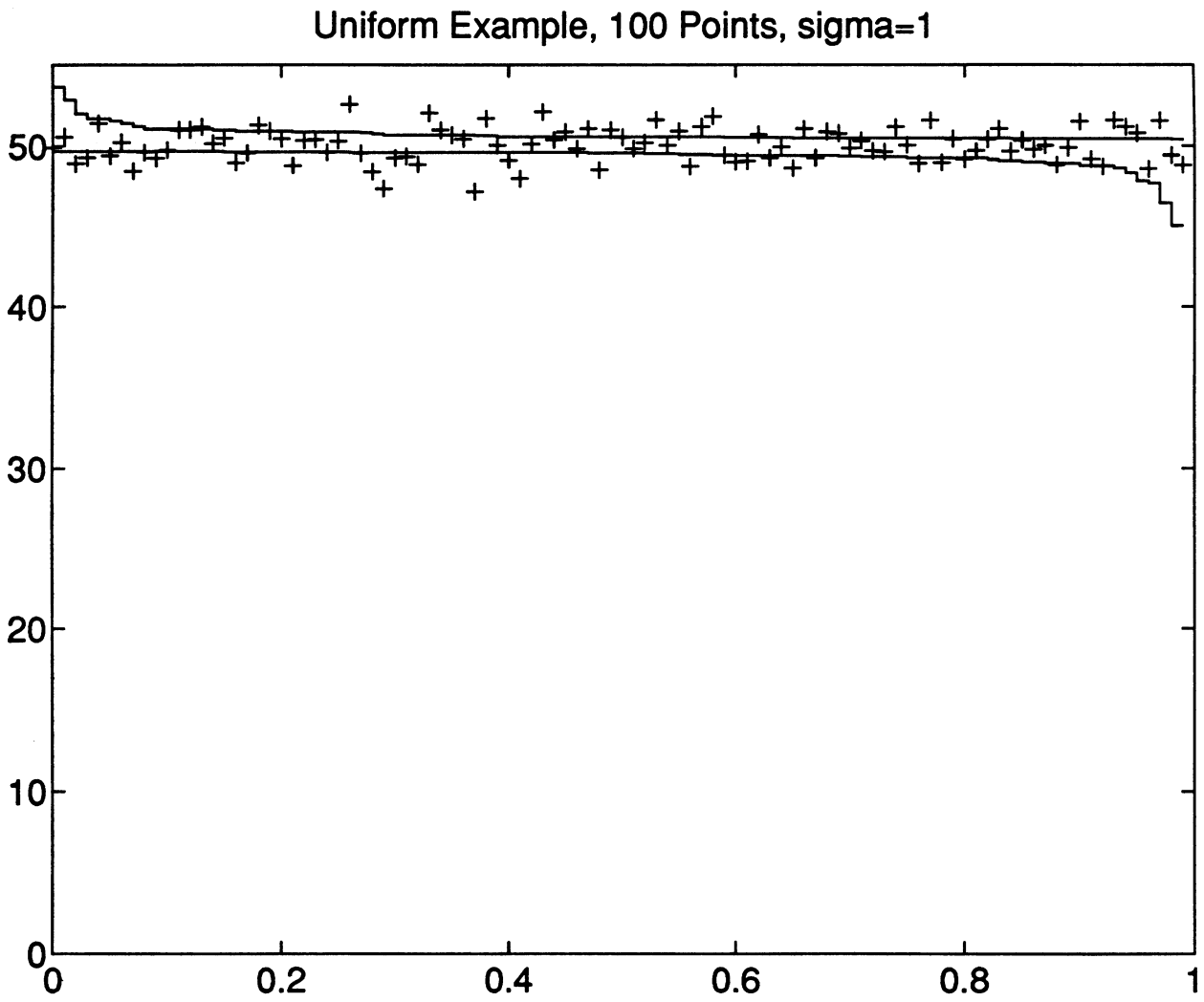

Cosine Example, 100 Points, sigma=1
120
100
80
60
40
20
0
-20
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6

Figure 2


Uniform Example, 100 Points, sigma=1
50
40
30
20
10
0
0
0.2
0.4
0.6
0.8
1