SOLVING LINEAR INEQUALITIES IN A LEAST SQUARES SENSE

R. BRAMLEY AND B. WINNICKA *

Abstract. In 1980, Han [5] described a finitely terminating algorithm for solving a system $Ax \leq b$ of linear inequalities in a least squares sense. The algorithm requires a singular value decomposition of a submatrix of A on each iteration, making it impractical for all but the smallest problems. This paper shows that a modification of Han's algorithm allows the iterates to be computed using QR factorization with column pivoting, which significantly reduces the computational cost and allows efficient updating/downdating techniques to be used. The effectiveness of this modification is demonstrated, implementation details are given, and the behaviour of the algorithm discussed. Theoretical and numerical results are shown from the application of the algorithm for linear separability problems.

1. Introduction. Let $A \in \Re^{m \times n}$ be an arbitrary real matrix, and let $b \in \Re^m$ a given vector. A familiar problem in computational linear algebra is to solve the system $Ax = b$ in a least squares sense; that is, to find an x^* minimizing $||Ax - b||$. Here and throughout this paper, $\| \cdot \|$ refers to the two-norm. Such an x^* solves the *normal equations* $A^T(Ax - b) = 0$, and the optimal residual $r^* = b - Ax^*$ is unique (although x^* need not be). The least squares problem is usually interpreted as corresponding to multiple observations, represented by the rows of A and b, on a vector of data x. The observations may be inconsistent, and in this case a solution is sought that minimizes the norm of the residuals. More information about linear least squares problems and solution techniques can be found in [7, 13, 3].

A less familiar problem to numerical linear algebraists is to solve systems of linear *inequalities* $Ax \leq b$ in a least squares sense, but the motivation is similar: if a set of observations places upper or lower bounds on linear combinations of variables, we want to find x^* minimizing $||(Ax - b)_+||$, where the i^{th} component of the vector v_+ is the maximum of zero and the i^{th} component of v. However, the algorithm has potential applications beyond simple data analysis, and later we show its use for linear separability problems. For that application, we want to find a hyperplane that best separates two point sets; when the two sets are not linearly separable, a hyperplane that correctly separates the largest number of points is desired.

When the system $Ax \leq b$ is consistent, that is, when a solution exists that satisfies all the inequalities, then phase I of any standard linear programming method can find it. Futhermore when the system is not consistent, linear programming can identify that case, but does not provide any kind of an "optimal" solution. Other methods

* Work supported by NSF grant CCR-9120105

developed for solving linear inequalities include an unusual algorithm by Stewart [12], which defines a function that diverges in a direction that converges to a solution of the inequalities; if no solution exists, the function converges to a unique minimum.

One way of solving the problem is to state it as the quadratic programming problem in (x, z)

$$(\mathrm{QP}) = \begin{cases} \min \frac{1}{2} z^T z \\ \text{subject to } Ax - b \leq z \end{cases} \tag{1}$$

However, there are serious numerical difficulties with solving a quadratic programming problem that has a singular objective function; furthermore, most methods require an active set strategy that can be difficult to implement, particularly when it is necessary to decide which entries to drop from the active set. As we will show, the analogue of an active set for the algorithm described in this paper is automatically determined without difficult decisions of when to drop a constraint.

The only algorithm specifically designed for solving linear inequalities in a least squares sense was developed by S.–P. Han [5]. That algorithm requires finding the minimum norm least squares (equality) solution to systems $A_I x = b_I$, where A_I is a submatrix of A consisting of rows of A. This implies that a singular value decomposition or a complete orthogonal decomposition of A_I is required on every iteration. Both of these decompositions are relatively expensive to compute, and there currently are no effective update/downdate methods that allow the reuse of work performed on a previous iteration. This paper will show that a minor modification of Han's algorithm allows an implementation using a QR factorization with column pivoting instead, and both the robustness and finite termination of Han's algorithm are retained.

Section 2 of this paper defines notation and reviews some basic properties of least squares solutions for linear inequalities, most of which can be found in [5]. Section 3 states Han's algorithm and shows the minor change in the convergence proof that allows QR with column pivoting to be used. Section 4 discusses implementation details and demonstrates the robustness of the modified algorithm. Section 5 shows the behaviour of the algorithm on selected (artificial) problems, and Section 6 compares the algorithm to a linear programming method for the linear separability problem.

2. Basics of systems of linear inequalities. This Section summarizes some fundamental properties of linear inequalities from Han's technical report, and proofs of the results can be found in [5]. Let $A \in \Re^{m \times n}$ be an arbitrary real matrix, and let $b \in \Re^m$ a given vector. No relation is assumed between m and n, and the matrix A can be rank-deficient, ill-conditioned, or even the zero matrix. Let the rows of A be denoted a_i^T, $i = 1, \ldots, m$. We want to find an $x \in \Re^n$ solving the system

$$Ax \leq b \tag{2}$$

in some sense. System (2) is interpreted componentwise, so we want $a_i^T x \leq b_i$ for all $i = 1, \ldots, m$. Note that this notation differs from that used by Mangasarian [9], where $\leq$ means at least one component satisfies the inequality strictly, that is, $a_k^T x < b_k$ for some k.

Possibly no such x exists. As an example, consider the system

$$A = \begin{bmatrix} 1 & 0 \\ -1 & 0 \end{bmatrix}, b = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \tag{3}$$

which is equivalent to $x_1 \leq 1$ and $x_1 \geq 2$. In these cases, we want a *least squares* solution, which we now define. Given a vector $v \in \Re^l$, define its positive part as the vector $v_+ \in \Re^l$ with components given by $v_+^i = \max\{0, v^i\}$. A least squares solution to (2) is any vector x that minimizes

$$f(x) = \frac{1}{2}\|(Ax - b)_+\|^2. \tag{4}$$

Note that analogous to the linear equality case, we can also define the (necessarily unique) x of minimum norm that solves $x = \operatorname{argmin} \|(Ax - b)_+\|$, but the method analyzed here does not provide a minimum norm solution.

Note that the function $f(x)$ is convex, continuously differentiable, and piecewise quadratic. Differentiating gives the analogue of the normal equations:

PROPOSITION 2.1. *$x^* \in \Re^n$ solves (2) if and only if $A^T(Ax^* - b)_+ = 0$.*

The proposition follows immediately from the convexity of f and the relation $\nabla f(x) = A^T(Ax - b)_+$.

Keeping in mind the similarity of Proposition 2.1 and the normal equations for equality linear least squares problems, we define the residual vector for (2) as

$$z = (Ax - b)_+. \tag{5}$$

This residual is zero if and only if the system of inequalities is consistent and x is a solution. Furthermore, by noting the equivalence of (2) and the quadratic programming problem (1), which is convex, we have

PROPOSITION 2.2. *For any matrix $A \in \Re^{m \times n}$ and vector $b \in \Re^n$, a least squares solution to (2) exists. The optimal residual vector $z^* = (Ax^* - b)_+$ is unique, and x is a least squares solution if and only if $(Ax - b)_+ = z^*$.*

Finally, we note that the gradient of f is globally Lipschitz of order 1, with a Lipschitz constant of $\|A\|^2$:

PROPOSITION 2.3. *$\|\nabla f(x) - \nabla f(y)\| \leq \|A\|^2 \|x - y\|$, for all $x, y \in \Re^n$.*

Again the proof is straightforward, using the relation $\nabla f(x) = A^T(Ax^* - b)_+$.

3. Modification of Han's algorithm. Two notations are needed for the statement of Han's algorithm. First, $G^\dagger$ denotes the pseudo-inverse of the matrix G [11]. In practice, all that is needed is the action of $G^\dagger$ on a vector f, not the linear operator itself in explicit form. The vector $G^\dagger f$ is the minimum norm, least squares solution to the problem of minimizing $||Gy - f||$, and can be computed by a QR factorization when G is full-rank, or by the singular value decomposition (SVD) otherwise. See Chapter 5 of [3] for details on computing these factorizations.

Second, let $I \subseteq \{1, 2, \ldots, m\}$ be an index set. Then A_I is the submatrix of A consisting of rows with indices in I. With this definition, $A_I \in \Re^{|I| \times n}$, where $|I|$ is the cardinality of I. The vector b_I can be defined similarly. Using this notation, the algorithm is:

Given: $A \in \Re^{m \times n}$, $b \in \Re^m$, a starting point $x^0 \in \Re^n$, and a tolerance $\epsilon > 0$.

Initialize:

- Set $k = 0$ (Iteration number)
- Set $r^0 = b - Ax^0$
- Set $I = \{i : a_i^T x^0 \geq b_i\} = \{i : r_i^0 \leq 0\}$ (Set of active indices)
- Set $\rho = ||r_I^0||$ (initial residual norm)

Iterate: While $(\rho > \epsilon)$

- $d^k = A_I^\dagger r_I^k$
- $\lambda = \text{argmin } f(x^k + \lambda d^k)$, where $f(x)$ is defined by equation (4).
- $x^{k+1} = x^k + \lambda d^k$
- $r^{k+1} = b - Ax^{k+1}$
- $I = I_{k+1} = \{i : a_i^T x^{k+1} \geq b_i\}$ (New set of active indices)
- $\rho^k = ||(r^{k+1})_I||$
- $k = k + 1$

Superscripts in the above algorithm indicate the iteration number. Note that I and λ depend on k also (otherwise the whole problem is trivial!), but for clarity, we omit the k when examining a single step of the algorithm. Note also that the exact line search specified for finding λ is reasonable in this case, since $\theta(\lambda) = f(x^k + \lambda d^k)$ is piecewise quadratic, convex, and continuous; we can simply search through the knot points to isolate an interval on which $\theta(\lambda)$ is quadratic, then interpolate. Furthermore, evaluating the line search function at a given point requires only matrix–vector products with A, a computationally efficient task. Numerical results presented later will show that the algorithm is in fact sensitive to the line search procedure, and it is worthwhile to consider other methods.

The fundamental result Han established about this algorithm is that it converges in

a finite number of steps to some minimizer of (4). The proof relies on the following properties, which are readily verified:

PROPOSITION 3.1. *For any matrix $A \in \Re^{m \times n}$ and vector $b \in \Re^n$, let $f(x)$ be given by (4), $x = x^k$, $d = d^k$, and $I = I(x^k)$. Then*

1. $\nabla f(x) = -A_I^T A_I d$
2. $d^T \nabla f(x) = -d^T A_I^T A_I d$
3. $d^T \nabla f(x) = -d^T A_I^T A_I d$, *so d is a descent direction for $f(x)$*
4. *d is the minimum norm least squares solution to the problem: minimize $\|A_I d - r_I\|$*

To make this algorithm computationally feasible, we want to compute d^k by means of a cheaper factorization of A_I. Omitting the superscript k temporarily, let $d_{\text{svd}} = A_I^\dagger r_I$. The general least squares solution to $A_I d = r_I$ is given by

$$d = d_{\text{svd}} + (I - A_I^\dagger A_I) y \tag{6}$$

for some vector $y \in \Re^n$. Note that the vector $(I - A_I^\dagger A_I)y$ is in the null space of A_I, so that $A_I d = A_I d_{\text{svd}}$. This is critical since, with one exception, the proof of finite termination in [5] involves only the quantity $A_I d$, not d. Before addressing that exception, we recall a result of Golub and Pereya [4]:

THEOREM 3.2. *Let G be an $m \times n$ matrix, and let $GP = Q\hat{R}$ be the QR with column pivoting factorization of G. Partition*

$$\hat{R} = \begin{bmatrix} R_1 & R_2 \\ 0 & 0 \end{bmatrix}, \tag{7}$$

where R_1 is $r \times r$, upper triangular, and invertible, and R_2 is $r \times n - r$. Let y_{svd} be the minimum norm least squares solution to the problem: minimize $\|Gy - f\|$, let $Q^T f = (c^T, g^T)^T$, where $c \in \Re^r$, and set

$$y = P \begin{pmatrix} R_1^{-1} c \\ 0 \end{pmatrix} \tag{8}$$

Then $\|y\|^2 \leq (1 + \|R_1^{-1} R_2\|^2)\|y_{\text{svd}}\|^2$

The application of this result to the linear inequality algorithm tells us that using QR factorization with column pivoting to compute the search direction d will always provide a d that is bounded by a constant times the norm of d_{svd}, where the constant does not depend on x or the iteration number k:

THEOREM 3.3. *Let d be computed in Han's algorithm using QR factorization with column pivoting. Then there is $C_R \geq 0$ such that $\|d\| \leq C_R \|d_{\text{svd}}\|$ independently of the iteration number k.*

Proof. For a given index set I, let R_1 and R_2 be the factors defined by applying Theorem 3.2 to the problem: miminize $\|A_I d - r_I\|$. Let $C_I = \sqrt{(1 + \|R_1^{-1} R_2\|^2)}$, and let $C_R = \max\{C_I\}$, where the maximum is taken over all possible index sets I. Since the number of such index sets is finite, C_R is well-defined. The result follows immediately from the last Theorem. □

We now prove the main convergence result needed:

THEOREM 3.4. *Let the sequences of vectors x^k and d^k be computed using the algorithm above, with d^k computed using QR factorization with column pivoting on A_I. Then either the algorithm stops after a finite number of iterations, or*

$$\lim_{k \to \infty} \nabla f(x^k) = 0 \tag{9}$$

Proof. If the algorithm stops after a finite number of iterations there is nothing further to prove. Suppose it takes an infinite number of iterations. Then for all steps, A_I is a nonempty matrix, since $I = \phi$ if and only if x^k is a solution. Initially we drop the superscripts k, and consider one step of the algorithm. Since ∇f is continuous and globally Lipschitz with a Lipschitz constant of $\|A\|^2$, [10, Theorem 8.3.1, p. 254] gives

$$f(x + \lambda d) \leq f(x) + \lambda d^T \nabla f(x) + \frac{\|A\|^2}{2} (\lambda \|d\|)^2 . \tag{10}$$

As a function of λ, the right hand side of the above bound has a minimum at

$$\hat{\lambda} = \frac{-d^T \nabla f(x)}{\|A\|^2 \cdot \|d\|^2}. \tag{11}$$

Substituting this value of λ in (10) and using $d^T \nabla f(x) = -d^T A_I^T A_I d$, we get

$$\begin{aligned} f(x + \hat{\lambda} d) - f(x) &\leq -\frac{1}{2} \left[\frac{d^T \nabla f(x)}{\|A\| \cdot \|d\|} \right]^2 \\ &\leq -\frac{1}{2} \left[\frac{\|A_I d\|^2}{\|A\| \cdot \|d\|} \right]^2 \end{aligned} \tag{12}$$

Note that if the component of d that lies in the null space of A_I becomes arbitrarily large, the upper bound above can go to zero. However, from Theorem 3.3 $\|d| \leq C_R \|d_{\text{svd}}\|$ where C_R is independent of the iteration number. Since $d_{\text{svd}} \in$ range (A_I^T), $d_{\text{svd}} = A_I^\dagger A_I d$ and so

$$\|d\| \leq C_R \|A_I^\dagger A_I d_{\text{svd}}\| \leq C_R \|A_I^\dagger\| \cdot \|A_I d_{\text{svd}}\| \leq C \|A_I d_{\text{svd}}\|, \tag{13}$$

where $C = \left(\max_I \|A_I^\dagger\|\right) C_R$. Substituting this bound on d into (12) gives

$$f(x + \hat{\lambda} d) - f(x) \leq -\frac{1}{2C\|A\|^2} \|A_I d\|^2. \tag{14}$$

Since the stepsize λ is chosen by an exact line search,

$$f(x) - f(x + \lambda d) \geq f(x) - f(x + \hat{\lambda} d) \geq \frac{1}{2C\|A\|^2} \|A_I d\|^2. \tag{15}$$

The last inequality holds for all iterations, and since $f(x^k)$ is monotone decreasing and bounded below by zero,

$$\sum_{k=0}^{\infty} \left[f(x^k) - f(x^k + \lambda d^k) \right] \geq \frac{1}{2C\|A\|^2} \sum_{k=0}^{\infty} \|A_I d\|^2 \tag{16}$$

is a finite sum, so $A_{I^k} d^k \to 0$ as $k \to \infty$ and hence $\nabla f(x^k) = -A_{I^k}^T A_{I^k} d^k \to 0$ as $k \to \infty$. □

As noted above, the rest of Han's proofs still apply to the modified algorithm, since they only rely on $A_I d$, not d. It is worthwhile to compare those results with related ones. As early as 1965, Katznelson [6] established finite termination of an algorithm for solving piecewise linear systems of equations that arise in circuits. More recently Li and Swetits [8] have established finite termination for solving systems of the form $\Phi(x) = Q^T[(Ax + b)_+ + (Cx + d)] = 0$ (c.f. the gradient of $f(x)$). In both cases, they relied on the assumption that within each polyhedral set created by the hyperplanes $H_i = \{x : a_i^T x = b_i\}$, the gradient of Φ is nonsingular. This corresponds to the case where $A_I^T A_I$ is nonsingular for all index sets I. The key idea is that within each polyhedral set the function is quadratic, and so if the k^{th} iterate lands in the polyhedral set containing the (necessarily unique) minimum, Newton's iteration converges in one step. Since there are a finite number of such polyhedral sets, it is a matter of showing that if an infinite number of the iterates lie in a single polyhedral set, they must converge to a point in the set. The unicity of solutions to $\nabla\Phi(x)d = -\Phi$ allows this by assuring that the iterates remain bounded.

Han's method applies a Gauss–Newton approach to the same problem, and restricts the choice of search directions d^k to minimum norm solutions, in order to have zero growth in the null space of A_{I^k}. The key idea used in this paper is that some growth in that null space is allowed, provided that it is uniformly bounded over all of the polyhedral sets, that is, over all choices of index set I. This result is important because it allows applying the algorithm to large systems, and the recent development of efficient and reliable orthogonal factorization methods for sparse systems allows it to be applied to the kind of systems that frequently arise in applications. Furthermore, methods for computing QR factorization with some form of pivoting on parallel machines allow implementation on modern high performance computers.

4. Implementation Details. The algorithm described above has been implemented in Matlab[1] with several options. Firstly, the algorithm clearly can be applied to mixed systems of the form

$$\begin{aligned} A^i x &\leq b^i \\ A^e x &= b^e \end{aligned} \tag{17}$$

The only modification required is to include the rows of A^e in the active set on every iteration. To simplify notation, for the rest of this paper we will continue to simply discuss the system $Ax \leq b$, but the program allows for the more general problem in (17). Secondly, options are included (via a menu choice) for scaling the rows of the matrix A. Just as in equality least squares, this may change the solution vector; however, in some applications (such as the linear separability problem), such scaling can improve the quality of the overall solution vector sought.

Provisions are made for finding the search direction in three ways: the singular value decomposition, QR factorization with column pivoting, and QR factorization with updating and downdating of the factors. The latter is used whenever

1. A_I has more rows than columns and appears numerically to be of full rank, and
2. the number of flops (floating point operations) for the update/downdate process is less than that required to perform the full QR factorization anew, and
3. numerical difficulty is not encountered within the downdate procedure;

otherwise we revert to recomputing the QR factorization with column pivoting.

A strong advantage of using this algorithm instead of a quadratic programming method for (1) is that we define the active index set exactly as it is given in the statement of the algorithm; no tolerance is used, and the test is against exact zero when determining if an index is active or not. This should be contrasted with quadratic programming methods, which generally rely on an active set strategy. Although those strategies can add an index reliably, the decision of when to drop an index is much less straightforward and often relies on rank determination of submatrices, a numerically difficult problem. Furthermore, quadratic programming methods can fail on this problem because it is semidefinite, generating directions that make the minimum appear to be at infinity. In particular, the quadratic programming method of Matlab fails on all the problems we have tried, even though it was written to handle indefinite quadratic programs.

Not surprisingly, the number of iterations required by the algorithm is sensitive to the line search performed. The reason for this is obvious; if the minimum of the

[1] The MathWorks, Inc.

line search function is at a boundary between the polyhedral sets defined in the last Section, then stopping short or overshooting the boundary can cause an index to be present or missing on the following iteration, when it should not be. Furthermore, some geometric reasoning shows that having a minimum of the line search function on a boundary is in fact common. For this reason, we have implemented two line search methods: a search through the knot points followed by quadratic interpolation as described in the last Section and a binary search method. The second method uses an initial search interval of [0,2], and increases the upper bound b (if necessary) until the derivative of the line search function is positive at b. As the numerical results will show, even with a binary search using over 50 function evaluations per iteration, the number of floating point operations within the line search procedure is a small fraction of those spent in finding the search directions.

The Matlab codes are instrumented to keep track of the flops within each phase of the program. These flop counts are approximate but some spot tests have shown them to be accurate to within 1% of the actual flop counts obtained by hand. We have not compared timings, saving those for a later Fortran implementation. However, the operations involved in performing QR factorization are generally more favorable to high performance computers than SVD factorizations, allowing good data locality, higher level BLAS routines, and pipelining of the computations. This will be examined in detail later using the Fortran implementation.

The Matlab files used are available via anonymous ftp at cs.indiana.edu; the files are located in the directory pub/bramley/ineq, and will be made available in netlib.

5. Numerical Characteristics of the Algorithm. In this Section, we numerically test the effects of the modification of Han's algorithm, and apply the algorithm to simple problems with $n = 2$, in order to graphically show the behaviour of the code. Figures 1, 2, and 3 show the ratios of floating point operations and iterations for 200 random problems of orders 80×40, 40×80, and 400×15, respectively. The left–hand plots are the ratios of the number of flops using a SVD to the number of flops using QR with column pivoting, and the same for the number of iterations required. The right–hand plots show those two ratios when QR with column pivoting is used, but this time comparing the use of quadratic interpolation to a binary search in the line search algorithm. All four plots for a given problem size are based on the same 200 problems, and the two showing ratios of flops are based on the total flops required, not just those spent in finding the search direction or performing the line search. Figure 1 shows that using SVD to find the search direction requires 2–10 times more flops than using QR factorization. The number of iterations can change by a factor of 2 either way, but even when QR requires twice as many iterations, the total number of flops is smaller than that used by the SVD method. From the two plots on the right in Figure 1, using quadratic interpolation is not always cheaper

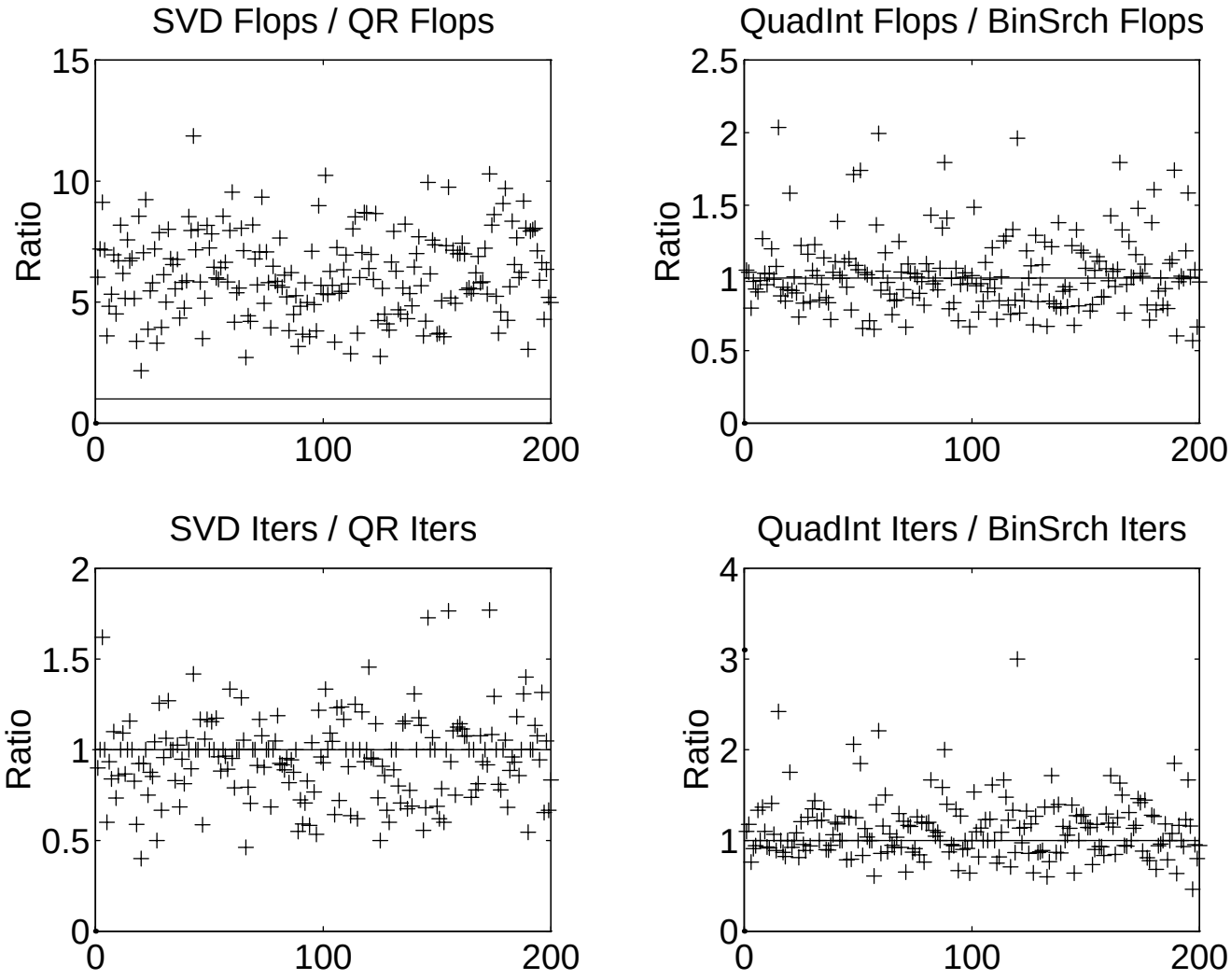

FIG. 1. *Ratios of Computational Costs for* 80×40 *Problems*

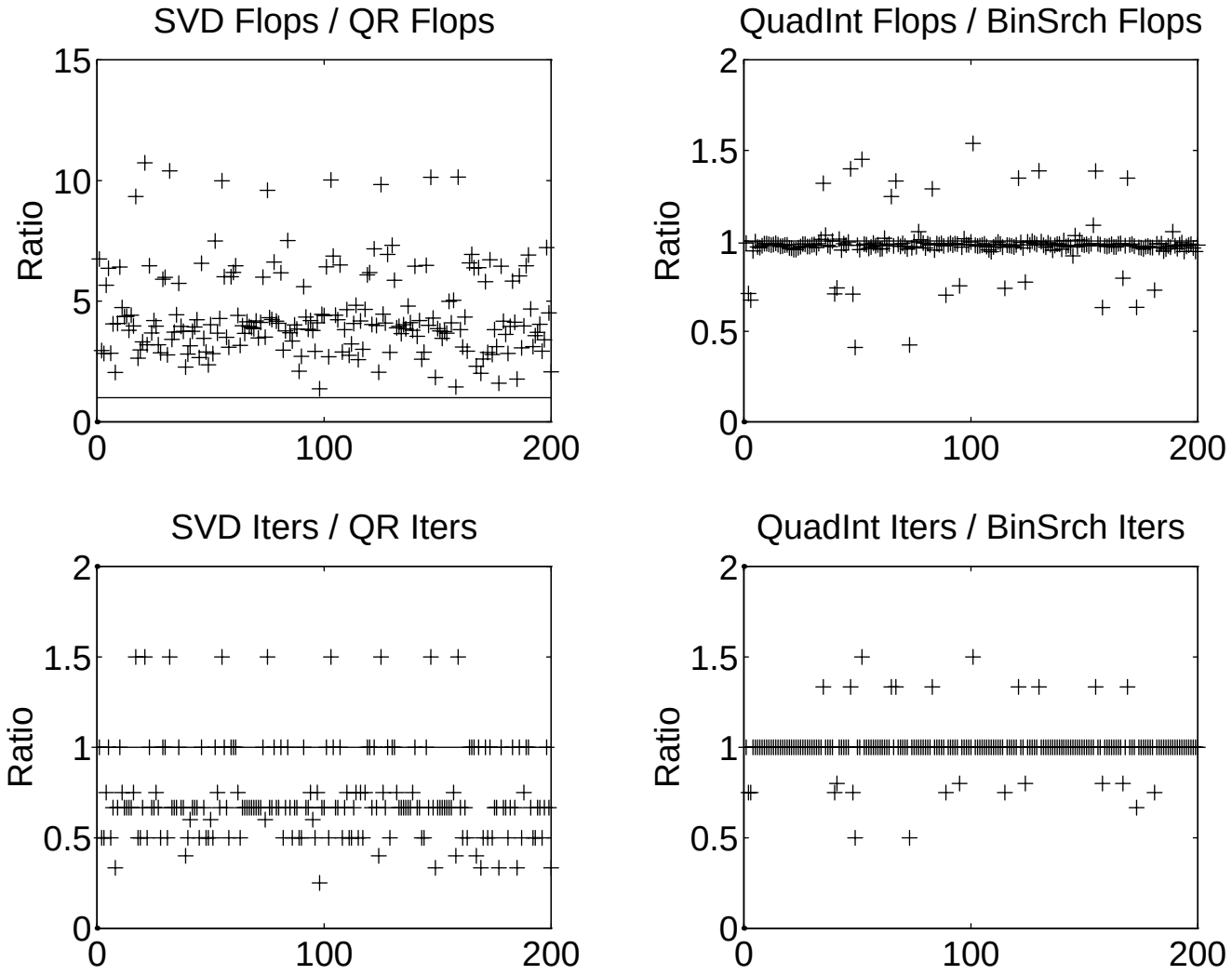

FIG. 2. *Ratios of Computational Costs for* 40×80 *Problems*

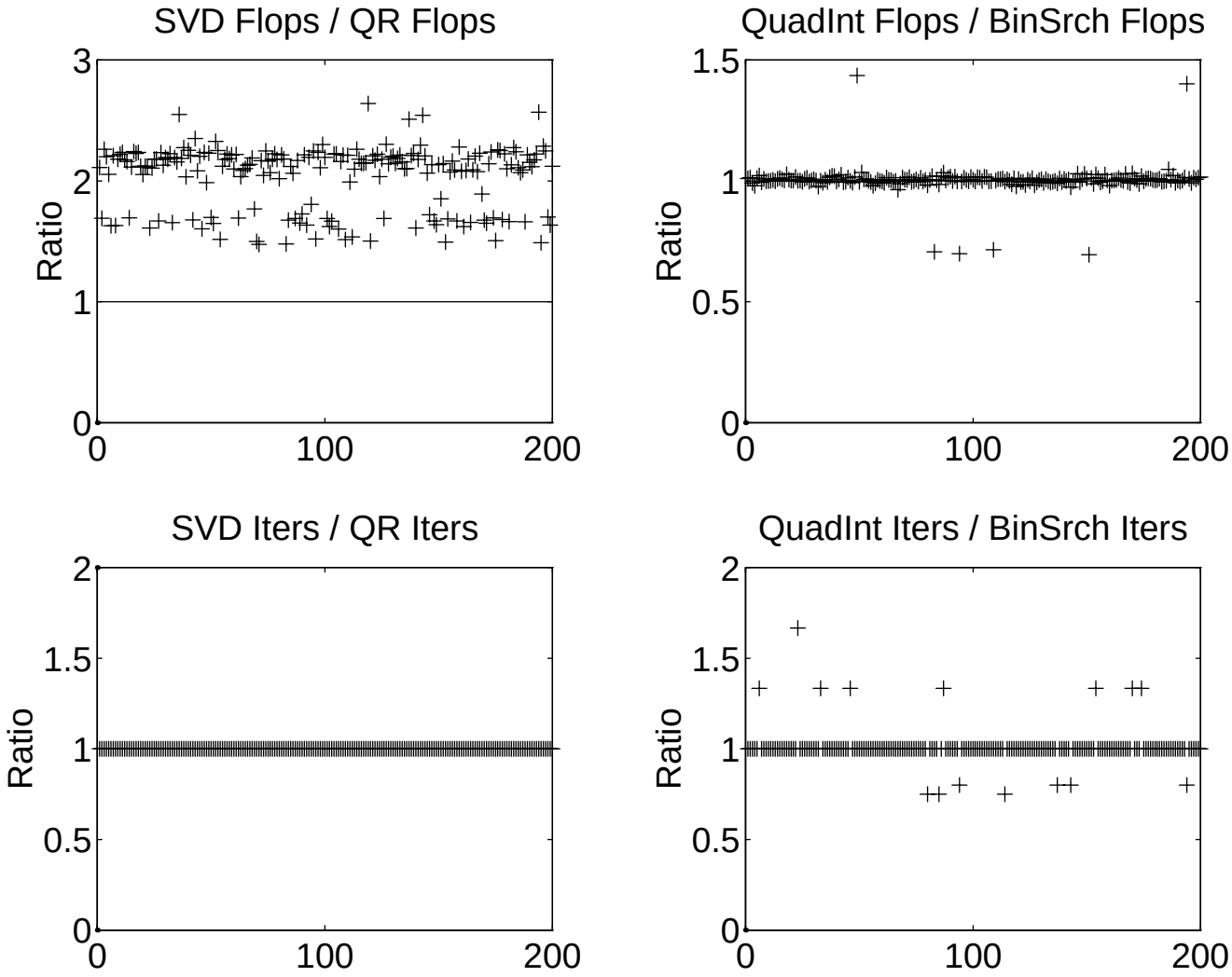

FIG. 3. *Ratios of Computational Costs for* 400×15 *Problems*

than using binary search, and can cause significantly more iterations.

Figure 2 shows that QR usually requires more iterations than using SVD. This makes intuitive sense, because for this problem size (40×80) many of the submatrices A_I don't have full column rank. Finding the search direction via QR with column pivoting can introduce some components from the null space of A_I, components that may have to be removed by later iterations. However, even when requiring 3 times more iterations, the QR–based method takes fewer flops; over 10 times fewer in some cases. The difference in the line search methods, however, is less significant for this underdetermined ($n \gg m$) case.

Finally, Figure 3 shows the results for problems with sizes similar to those used in Section 6. Again, the QR–based method is always cheaper, and requires 1.5–3 times fewer flops. The number of iterations does not change, however. For this problem size, A_I usually has more rows than columns and has full column rank; in this case the two methods for finding the search direction should generate the same direction. As with the 40×80 case, the choice of line search method has little impact.

The conclusions from these experiments are that using a binary line search method is cheaper than a quadratic interpolation as often as not, and altering the algorithm to select the search direction with a QR factorization is an important improvement. The improvement occurs in overdetermined ($m \gg n$) cases, and is large even when $n \gg m$, when the extra freedom introduced in null(A_I) could potentially cause numerical difficulties.

Next, we examine the behaviour of the algorithm for problems with $n = 2$, which allows graphical representation. Figure 4 shows the polyhedral sets for Problem 1: $Ax \leq b$, with

$$A = \begin{bmatrix} 0 & -1 \\ -1 & 0 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{3}{\sqrt{34}} & \frac{5}{\sqrt{34}} \end{bmatrix}, \quad b = \begin{pmatrix} -1 \\ -1 \\ \frac{1}{\sqrt{2}} \\ \frac{7}{2\sqrt{34}} \end{pmatrix}. \tag{18}$$

Problem 2 is shown in Figure 5, and is the same as Problem 1 but with -1 replace by 1 in (18). Overlain in both Figures are the contours of the function f, and arrows showing the search direction d corresponding to various points in the plane. The base of each arrow is at the corresponding point x from which it was calculated, and the length of the arrow is proportional to the length of the search direction d. Problem 1 has a unique solution x^*, in the interior of the large triangle. Also note that the contours look like those of a straightforward quadratic functional. However, the piecewise nature of the function can been seen where the vectors d change direction abruptly at hyperplane boundaries. Problem 2 is a consistent problem with any point

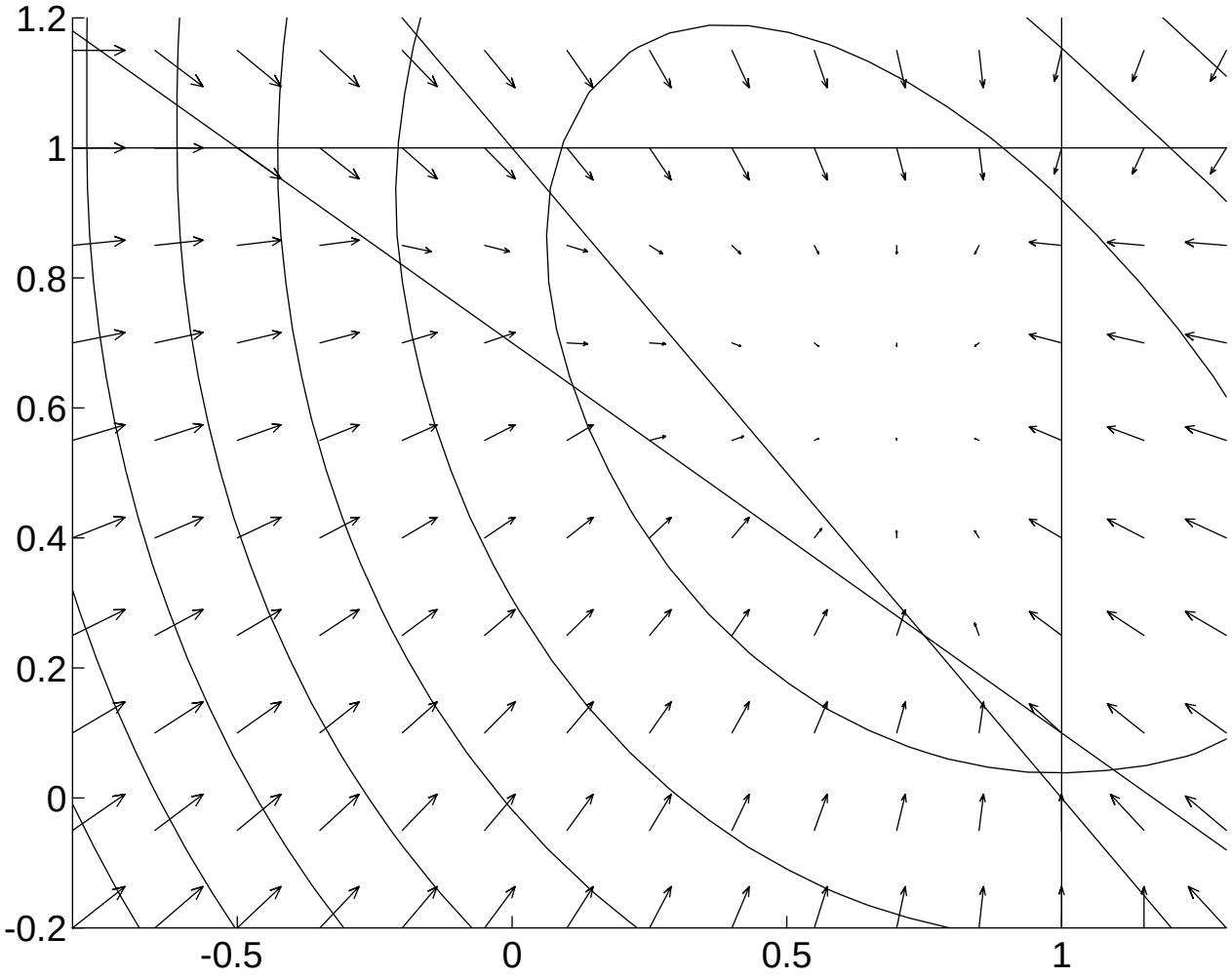

FIG. 4. *Flowfield for Algorithm for Problem 1*

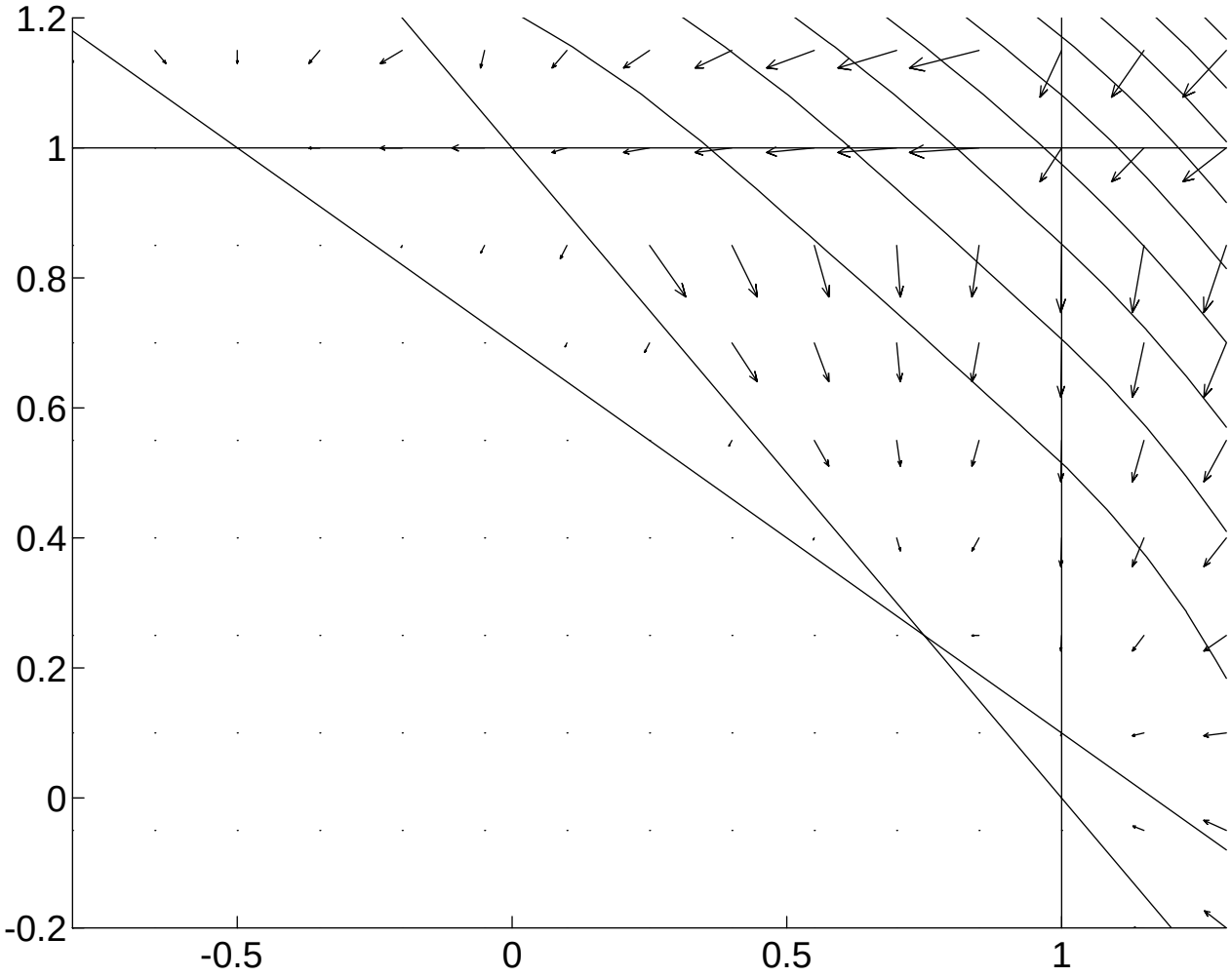

FIG. 5. *Flowfield for Algorithm for Problem 2*

in the large region to the lower left a least squares solution. Here the piecewise nature of the function is clear from the contours.

Next, summaries of results on random problems are given. Problems were randomly generated with entries for A and b being normally distributed with mean zero and variance 1 (the RANDN function of Matlab). Randomly generated problems can be misleading since, e.g., they are full rank with high probability. However, three interesting results come from randomly generated problems. Figure 6 shows the percentage of occurrences of the algorithm requiring k iterations for three problem sizes, for several hundreds of test cases. In no case has the method required more than $1 + \max(m, n)$ iterations. Furthermore, for most cases the number of required iterations is much smaller than this upper bound.

Figures 7 and 8 show surface and contour plots of the maximum number of iterations required for problem sizes ranging from 10×10 up to 200×200; these are more extensive versions of the table in Han's technical report. Note that the largest number of iterations occurs when $m \approx 2n$, which is the second observation that can be obtained from randomly generated problems.

Finally, when $m < n$, usually only 1–3 iterations are required. This makes heuristic sense because then the number of degrees of freedom exceeds the number of "constraints" imposed by the system.

6. Application to the linear separability problem. The linear separability problem is the one of finding a best hyperplane that separates two point sets $\mathcal{A}$ and $\mathcal{B}$ in $\Re^n$. Suppose there are m points in $\mathcal{A}$ and k points in $\mathcal{B}$. Let A and B be matrices with rows giving the coordinates of the points in $\mathcal{A}$ and $\mathcal{B}$, respectively. Then $A \in \Re^{m \times n}$ and $B \in \Re^{k \times n}$. We want to find $w \in \Re^n$ and a scalar γ so that $Aw \leq \gamma e_m$ and $Bw > \gamma e_k$, where $e_i \in \Re^i$ is a vector of all ones.

Clearly not all sets $\mathcal{A}$ and $\mathcal{B}$ can be separated by a hyperplane, so we want to find a hyperplane that is optimal in the sense of having fewest points incorrectly classified as belonging to $\mathcal{A}$ or $\mathcal{B}$. This can be approximated as the least squares inequality problem

$$\begin{aligned} Aw - \gamma e_m &\leq -e_m \\ -Bw + \gamma e_k &\leq -e_k \end{aligned} \tag{19}$$

Note that this formulation actually specifies that all the points lie outside of the "slab" $H = \{x : \gamma - 1 \leq w^T x \leq \gamma + 1\}$. This guards against the case when all the data points line up on a hyperplane, in which case formulating the least squares problem (19) with the zero vector on the right hand side would give the trivial solution $(w, \gamma) = 0$. Also

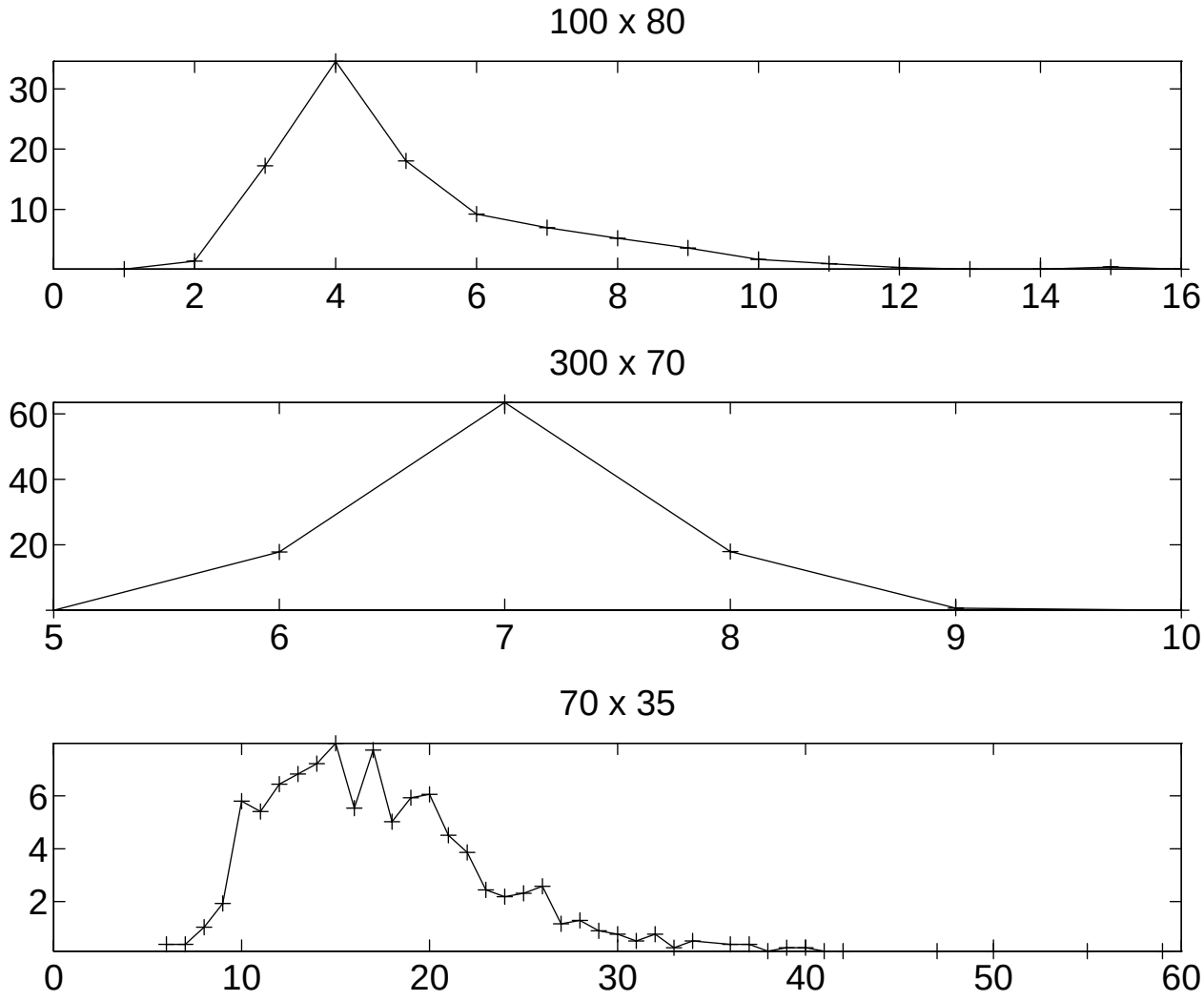

FIG. 6. *Percentage of Problems Requiring k Iterations for Three Problem Sizes*

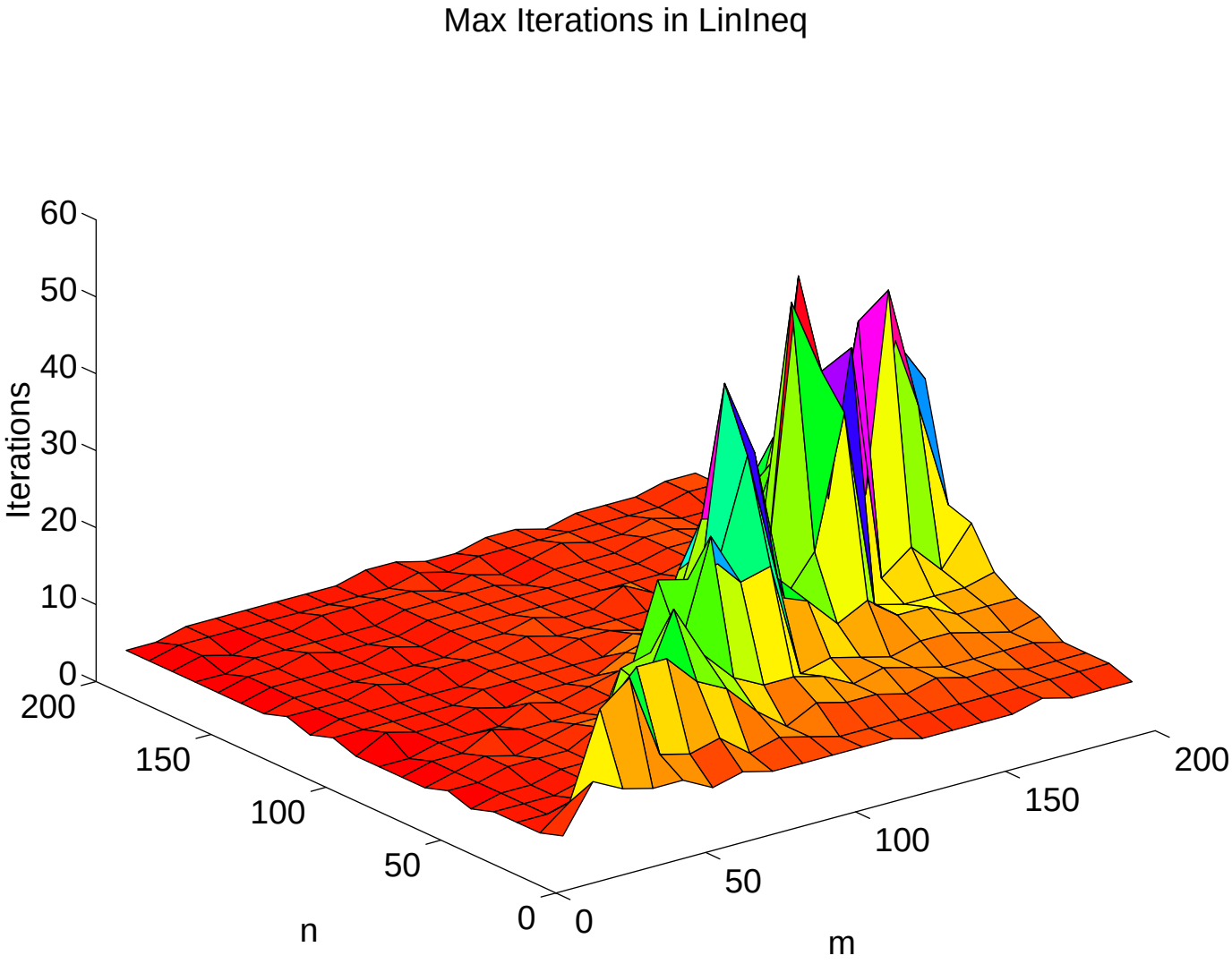

FIG. 7. *Maximum Iterations Required for $m \times n$ Problems*

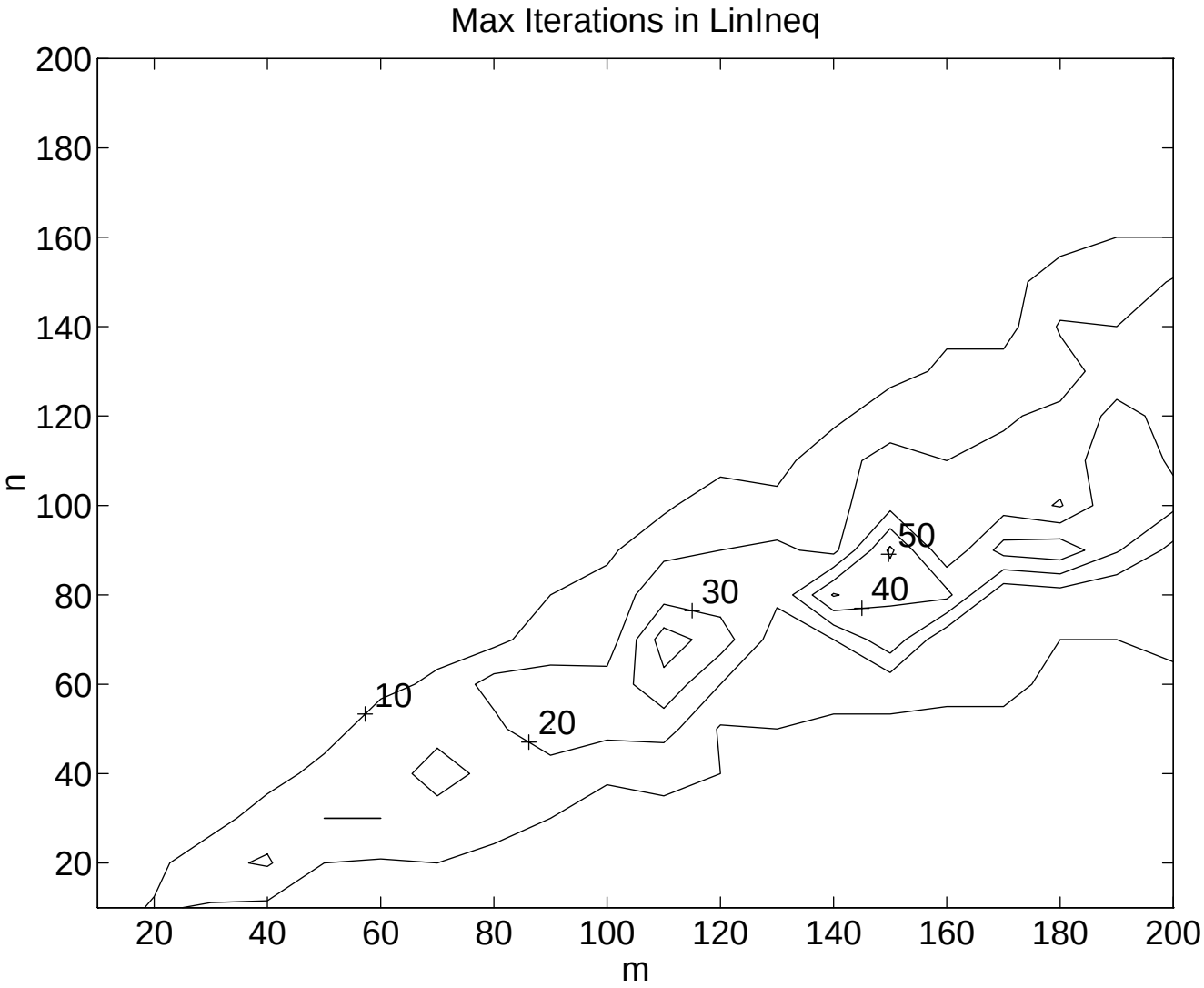

FIG. 8. *Maximum Iterations Required for $m \times n$ Problems*

note that the use of the constant 1 is arbitrary, since we may scale the vector (w, γ) without changing the placement or orientation of the resulting hyperplane.

In [1], Bennett and Mangasarian formulate this problem in terms of the $\mathcal{L}_1$ norm: miminize

$$\frac{1}{m}\sum_{i=1}^{m}\left(-a_i^T w + \gamma + 1\right)_+ + \frac{1}{k}\sum_{j=1}^{k}\left(b_j^T w - \gamma + 1\right)_+ \tag{20}$$

and then solve it as the linear programming problem: minimize over (w, γ, y, z)

$$\frac{y^T e_m}{m} + \frac{z^T e_k}{k} \tag{21}$$

subject to the constraints

$$\begin{aligned} Aw - \gamma e_m + y &\geq e_m \\ -Bw + \gamma e_k + z &\geq e_k \\ y, z &\geq 0 \end{aligned} \tag{22}$$

The normalization of terms of the objective function by $1/m$ and $1/k$ assures that nontrivial solutions (w, γ) exist. A similar result holds for the least squares formulation (19):

THEOREM 6.1. *Suppose the two point sets $\mathcal{A}$ and $\mathcal{B}$ have m and k points, respectively, and $m, k > 0$. Then the least squares formulation of (19) has the trivial solution $w = 0$ if and only if*

$$\sum_{i=1}^{m} a_i = \alpha \sum_{j=1}^{k} b_j \tag{23}$$

for some constant α.

Proof. Suppose that the least squares solution of (19) has $w = 0$, let $\tilde{w} = (w^T, \gamma)^T$, and let

$$G = \begin{bmatrix} A & -e_m \\ -B & e_k \end{bmatrix}, \quad g = \begin{pmatrix} -e_m \\ -e_k \end{pmatrix}. \tag{24}$$

Then

$$\begin{aligned} \|(G\tilde{w} - g)_+\|^2 &= \left\| \begin{pmatrix} (1-\gamma)e_m \\ (1+\gamma)e_k \end{pmatrix}_+ \right\|^2 \\ &= \begin{cases} m(1-\gamma)^2, & \text{if } \gamma < -1 \\ k(1+\gamma)^2, & \text{if } \gamma > 1 \\ m(1-\gamma)^2 + k(1+\gamma)^2, & \text{if } -1 \leq \gamma \leq 1 \end{cases} \end{aligned} \tag{25}$$

Furthermore, the active index set I is

$$I = \begin{cases} \{1, 2, \ldots, m\}, & \gamma < -1 \\ \{m+1, m+2, \ldots, m+k\}, & \gamma > 1 \\ \{1, 2, \ldots, m+k\}, & -1 \leq \gamma \leq 1 \end{cases} \tag{26}$$

From the normal equations, $G^T(G\tilde{w} - g)_+ = 0$. If $\gamma < -1$, this implies

$$0 = \begin{pmatrix} A^T(1-\gamma)e_m \\ -m(1-\gamma) \end{pmatrix} = (1-\gamma) \begin{pmatrix} \sum_{i=1}^m a_i \\ -m \end{pmatrix} \tag{27}$$

and so $m = 0$, a contradiction. Similarly, if $\gamma > 1$, this implies

$$0 = \begin{pmatrix} -B^T(1+\gamma)e_k \\ k(1+\gamma) \end{pmatrix} = (1+\gamma) \begin{pmatrix} -\sum_{j=1}^k b_j \\ k \end{pmatrix} \tag{28}$$

and so $k = 0$, again a contradiction. So $-1 \leq \gamma \leq 1$, in which case

$$\begin{aligned} 0 &= \begin{pmatrix} A^T(1-\gamma)e_m - B^T(1+\gamma)e_k \\ -m(1-\gamma) + k(1+\gamma) \end{pmatrix} \\ &= \begin{pmatrix} (1-\gamma)\sum_{i=1}^m a_i - (1+\gamma)\sum_{j=1}^k b_j \\ \gamma(m+k) + (k-m) \end{pmatrix} \end{aligned}$$

If $|\gamma| = 1$, the last component implies that one of m or k is zero. So $|\gamma| < 1$, and the theorem holds with $\alpha = (1+\gamma)/(1-\gamma)$. □

Note that it is easy to check for the condition in (23) before beginning computations, and perturbing any entry of the matrix G will prevent the trivial solution from occuring. This should be contrasted with the $\mathcal{L}_1$ norm minimizing method of [1], where only the objective function for the linear program need be scaled. However, in that formulation it is only guaranteed that nontrivial solutions exist to the linear program, but it is not assured that a linear programming program will find such a nontrivial solution. In any case, the condition in (23) has so far only occured in artificially created problems.

6.1. Two-dimensional Tests. We have compared the quality of solution of both approaches for several linear separability problems. For point sets that are separable, both methods give correct solutions. For nonseparable point sets, it is useful to first consider problems in the plane. Two artificial problems were used: the first is a smaller square inside the unit square, and the second are two triangles that overlap. For both methods, after (w, γ) were found a secondary minimization on γ was performed to improve the solution. This one-dimensional minimization is easily carried out by searching through the knot points defined by $\gamma_i = a_i^T w$ and $\gamma_j = b_j^T w$

and adds little to the overall computation costs. Table 1 shows the number of points incorrectly classified by the two methods. For the triangles problem, both triangles have one hundred data points, while two hundred total data points were used for the square problem, with the smaller square having a number of points proportional to its area. Figures 9–10 show the resulting solutions. Both the hyperplane (the central

Method	Square Problem	Triangles Problem
$\mathcal{L}_1$	14	53
$\mathcal{L}_2$	11	52

TABLE 1
Number of Incorrectly Classified Points in 2D

line) and the flanking lines that define the slab are shown.

To see how the two approaches compare as the degree of inseparability increases, the amount by which the surface area of the two triangles overlap was varied and the percentage of incorrectly classified points found. The results are shown in Figure 13. As can be seen, both methods give comparable (and reasonable) solutions. However, for the least squares solution, there is a physical interpretation of the flanking hyperplanes that define the slab:

PROPOSITION 6.2. *Let (w, γ) be a least squares solution to the system (19). The sum of residuals corresponding to point set $\mathcal{A}$ equals the sum of residuals corresponding to point set $\mathcal{B}$.*

Proof. From the normal equations for (19),

$$\begin{bmatrix} A^T & -B^T \\ -e_m^T & e_k^T \end{bmatrix} \begin{pmatrix} Aw - (\gamma - 1)e_m \\ -Bw + (\gamma + 1)e_k \end{pmatrix}_+ = 0 \tag{29}$$

The last scalar equation from above gives

$$e_m^T(Aw - (\gamma - 1)e_m)_+ = e_k^T(-Bw + (\gamma + 1)e_k)_+ \tag{30}$$

or equivalently

$$\sum_{i=1}^{m} \left[a_i^T w - (\gamma - 1)\right]_+ = \sum_{j=1}^{k} \left[-b_i^T w + (\gamma + 1)\right]_+, \tag{31}$$

which is the statement of the proposition. □

The last proposition shows that the inequality least squares problem will translate the separating hyperplane to where the sum of residual violations is balanced; this is not a criterion of the original linear separability problem, and is the reason we follow

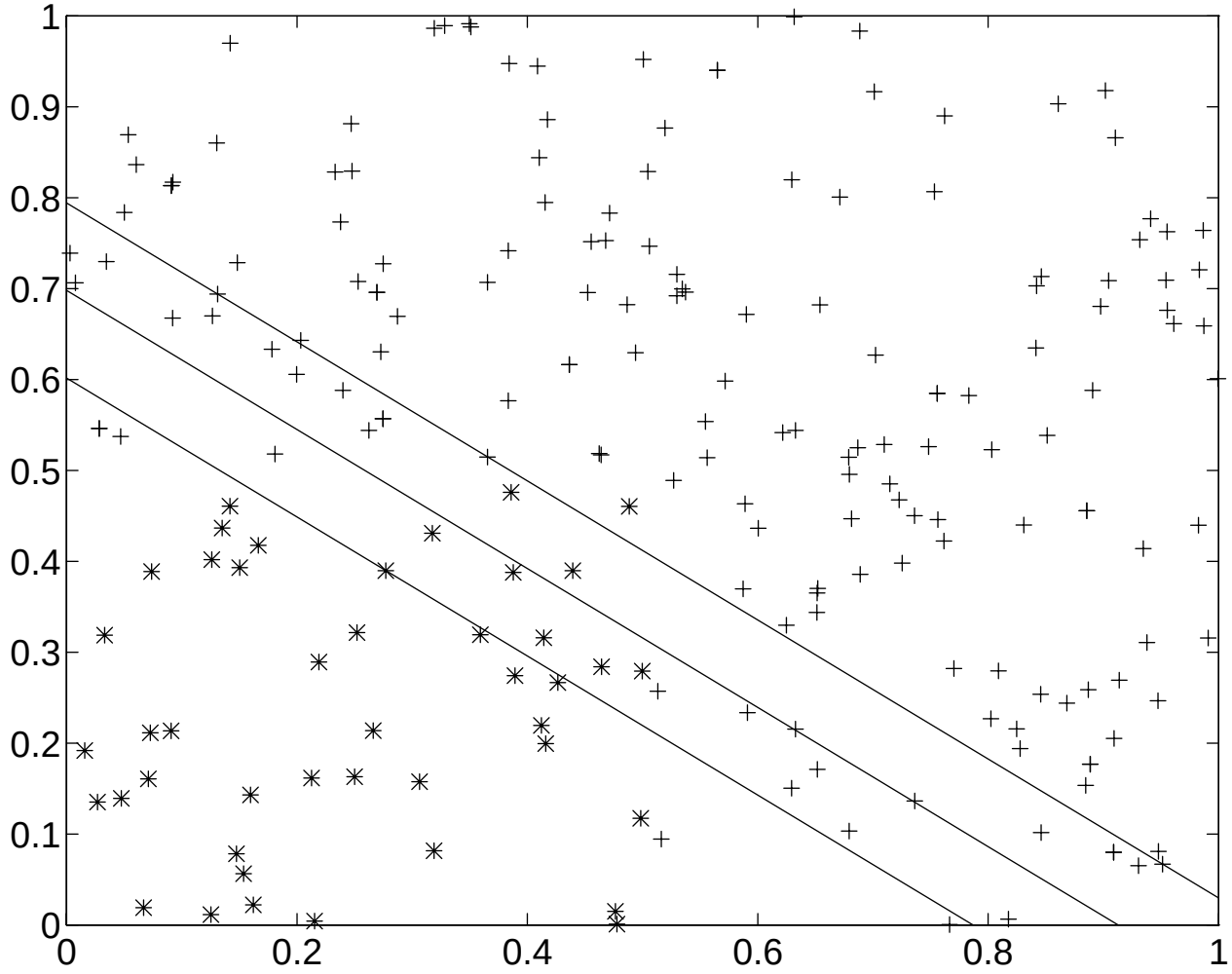

FIG. 9. *Linear Programming Solution to Squares Problem;* $\mathcal{A}: +$, $\mathcal{B}: *$

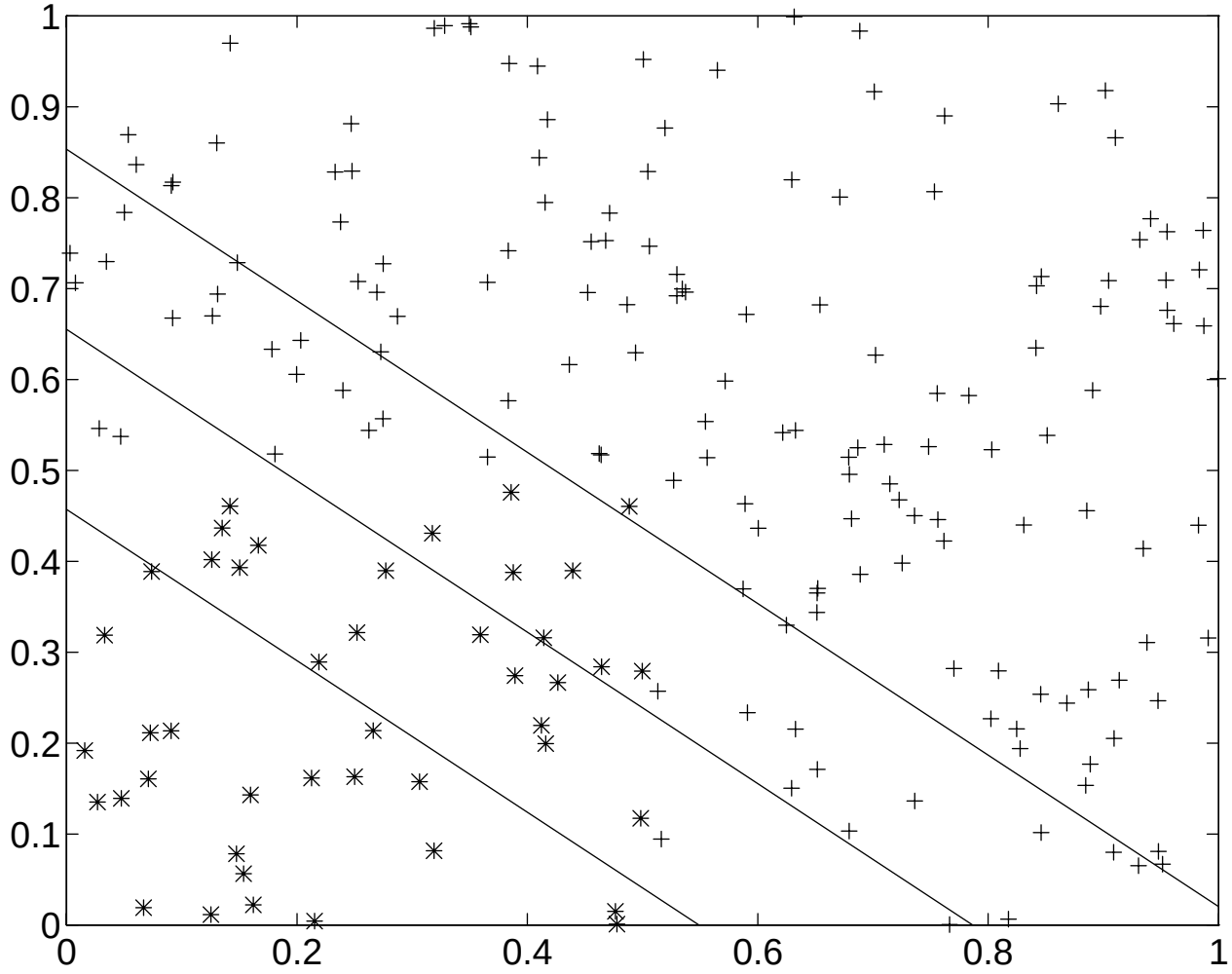

FIG. 10. *Inequality Least Squares Solution to Squares Problem;* $\mathcal{A} : +$, $\mathcal{B} : *$

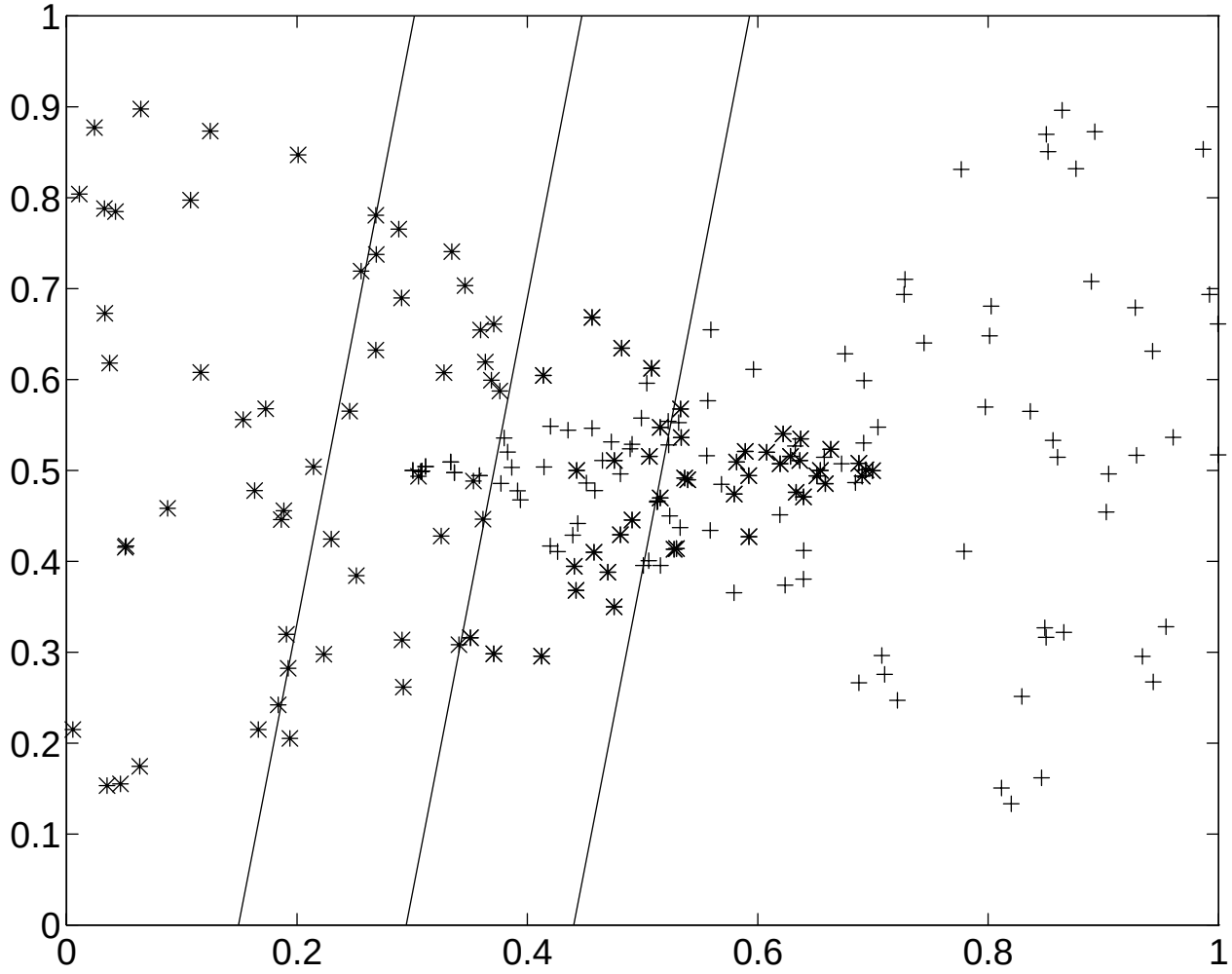

FIG. 11. *Linear Programming Solution to Triangles Problem;* $\mathcal{A}: +$, $\mathcal{B}: *$

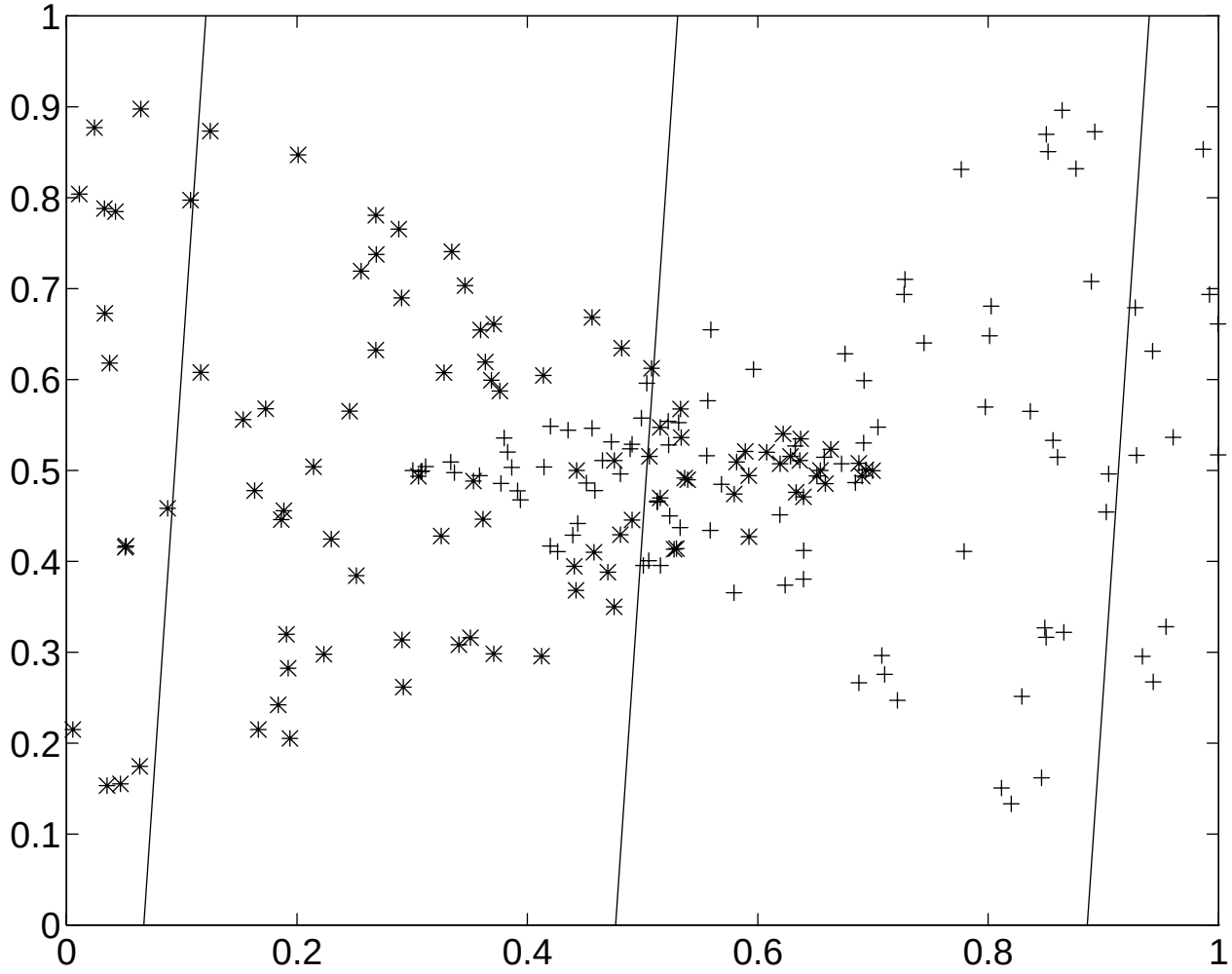

FIG. 12. *Inequality Least Squares Solution to Triangles Problem;* $\mathcal{A} : +$, $\mathcal{B} : *$

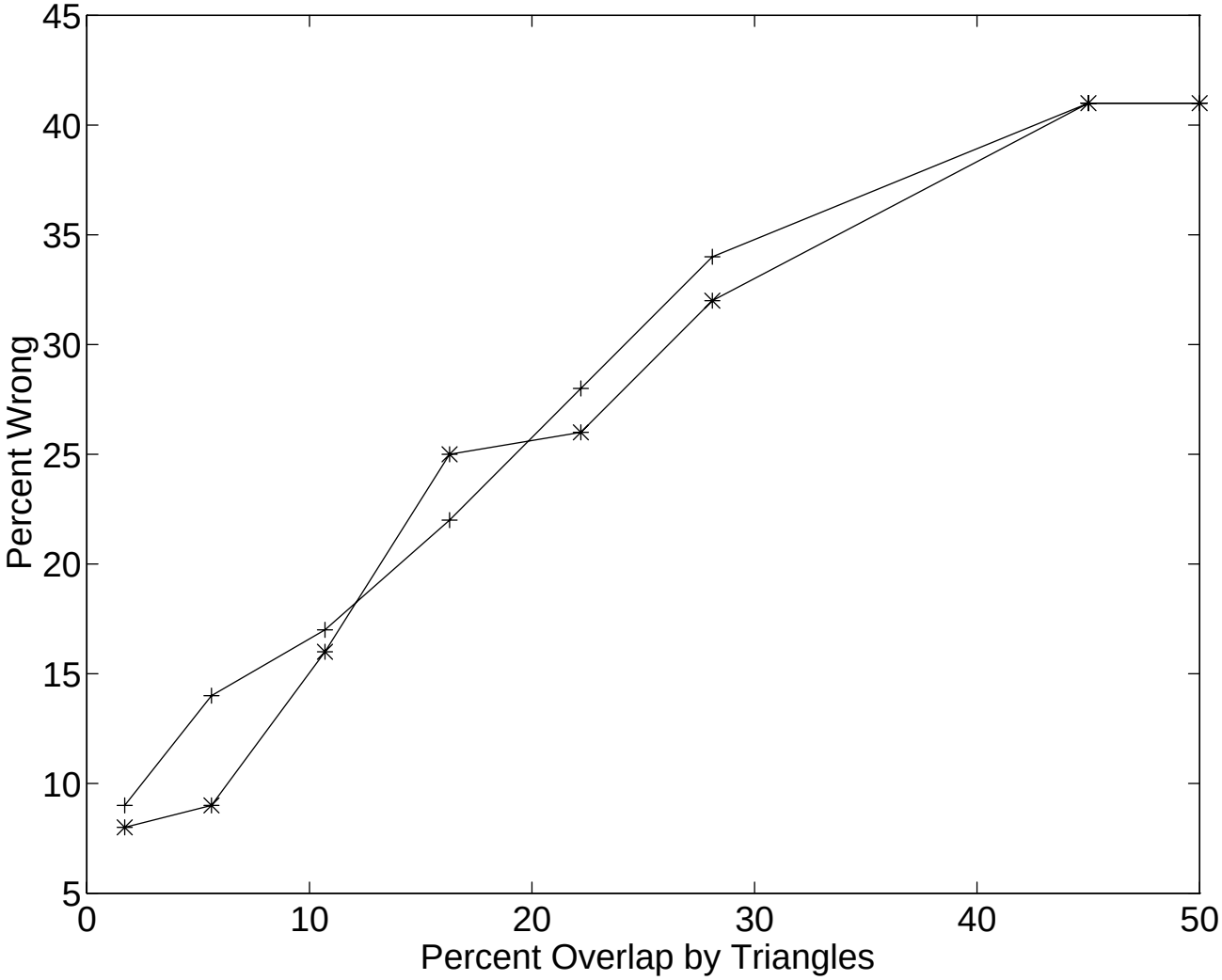

FIG. 13. *Percent Incorrect Points versus Percent Overlap for Two Triangles Problem*

the iterations to find (w, γ) with a one-dimensional minimization on γ, to translate the hyperplane to a more favorable position. This usually improves the solution by a few data points for two-dimensional test problems, but is much less effective for higher dimensional problems. Intuitively, in the two-dimensional case minimizing on γ varies one third of the variables, while for a 9 or 13 dimensional problem, only one tenth or fourteenth of the variables are being changed.

6.2. Performance on Test Databases. We have also run tests on the Wisconsin Breast Cancer Database and the Cleveland Heart Disease Database [2].

Both data sets are available from the University of California–Irvine Repository Of Machine Learning Databases and Domain Theories[2]. The first data set consists of 551 points, 346 from set $\mathcal{A}$ (corresponding to benign tumors) and 205 from set $\mathcal{B}$ (corresponding to malignant tumors). Each data point has 9 components, corresponding to experimental measurements. The second data set consists of 297 points, 137 from set $\mathcal{A}$ (corresponding to a negative diagnosis) and 160 from set $\mathcal{B}$ (corresponding to a positive diagnosis).

We discarded data samples that had missing measurements. As in [1], the data was divided randomly into a training group consisting of 2/3 of the data points, and a testing group consisting of the remaining 1/3 of the data. The best separating hyperplane was found by applying the inequality least squares solver using the training group of data, and then the effectiveness of the hyperplane was tested on the remaining testing group of data. This was repeated ten times, with different partitionings into training and testing sets. For the first data set, the inequality solver required 7 or 8 iterations each time and spent an average of 92% of its flops in finding the search direction. For the second data set, 5 or 6 iterations were needed each time and an average of 95% of the flops were spent in finding the search direction.

Table 2 shows the results both with and without the secondary minimization on γ, along with similar results from [1]. The results are not strictly comparable since that work used 566 data points from the Wisconsin Cancer Database and 197 data points from the Cleveland Heart Disease Database, but the comparison suggest that the inequality least squares method provides a solution similar to that of a linear programming method. This is unexpected. The measure of correctness used here is given by a simple count of the number of correctly classified points, and the two–norm solution given by inequality least squares will tend to heavily weight outlying data points. The one–norm solution given by the linear programming method weights outlying data points less than the two–norm solution does, and so would be expected to better model the true objective function.

[2] via anonymous ftp to ics.uci.edu, in directory pub/machine-learning-databases

Database	Data Group	Ls Sq w/o γ Min	Ls Sq w γ Min	Lin Progr
Cancer	Training	3.19	2.64	3.01
	Testing	4.24	3.80	2.56
Heart	Training	14.75	13.84	15.23
	Testing	15.76	15.96	16.53

TABLE 2
Percent of Incorrectly Classified Points For Two Database

7. Summary and Future Work. A computationally efficient implementation of Han's algorithm for solving linear inequalities in a least squares sense has been presented, and has been shown convergent by minor modifications to his convergence proofs. The effectiveness of this change in the algorithm's implementation has been demonstrated, and it has been tested on illustrative two-dimensional problems, randomly generated problems, and linear separability problems.

The most interesting work remaining is a proof of convergence within $\max(m, n) + 1$ iterations, or a counterexample. Since QR factorization with column pivoting takes $\mathcal{O}(mn^2)$ work in the dense case, this would provide a polynomial upper bound on the algorithm. A linear programming problem can be stated as a system of inequalities [5], so this algorithm would be another polynomial method for linear programming. Furthermore, the subproblems generated by the inequality least squares algorithm have condition numbers no worse than that of the original data, since at each step a decomposition is performed on a submatrix of A. Most interior point methods for linear programming have subproblems that become increasingly ill-conditioned as the iterates converge. So if the conjectured upper bound on the number of iterations holds, this algorithm provides a numerically stable polynomial time algorithm for linear programming.

REFERENCES

[1] K. BENNETT AND O. MANGASARIAN, *Robust linear programming discrimination of two linearly inseparable sets*, Optimization Methods and Software, 1 (1992), pp. 23–34.

[2] R. D. ET AL, *International application of a new probability algorithm for the diagnosis of coronary artery disease*, American Journal of Cardiology, 64 (1989), pp. 304–310.

[3] G. GOLUB AND C. V. LOAN, *Matrix Computations*, John Hopkins University Press, Baltimore, 2 ed., 1989.

[4] G. GOLUB AND V. PEREYSA, *Differentiation of pseudoinverse, separable nonlinear least squares problems and other tales*, in Generalized Inverses and Applications, M. Nashed, ed., Academic Press, New York, 1976, pp. 303–323.

[5] S.-P. HAN, *Least-squares solution of linear inequalities*, Tech. Rep. TR–2141, Mathematics Research Center, University of Wisconsin-Madison, 1980.

[6] S. KATZNELSON, *An algorithm for solving nonlinear resistor networks*, Bell System Technical Journal, 44 (1965), pp. 1605–1620.

[7] C. L. LAWSON AND R. J. HANSON, *Solving Least Squares Problems*, Prentice–Hall, Englewood Cliffs, NJ, 1974.
[8] W. LI AND J. SWETITS, *A Newton method for solving convex quadratic programs*, SIAM Journal on Optimization, 3 (1993), pp. 466–488.
[9] O. L. MANGASARIAN, *Nonlinear Programming*, McGraw–Hill Book Co., New York, 1969.
[10] J. M. ORTEGA AND W. C. RHEINBOLDT, *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press, New York, 1970.
[11] R. PENROSE, *A generalized inverse for matrices*, Proc. Cambridge Phil. Soc., 51 (1955), pp. 406–413.
[12] G. STEWART, *An iterative method for solving linear inequalities*, Tech. Rep. TR–1833, University of Maryland Computer Science Department, 1987.
[13] G. W. STEWART, *Introduction to Matrix Computations*, Academic Press, New York, 1973.