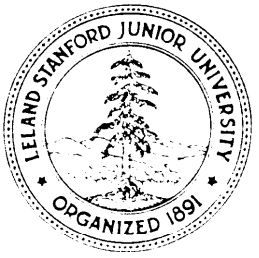
A GENERALIZED CONJUGATE GRADIENT ALGOR ITHM FOR SOLVING A CLASS OF QUADRATIC PROGRAMMING PROBLEMS

by

Dianne Prost O'Leary

STAN-CS-77-638
DECEMBER 1977

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY



Su326 P30-57

A GENERALIZED CONJUGATE GRADIENT ALGORITHM FOR SOLVING A CLASS OF QUADRATIC PROGRAMMING PROBLEMS

by

Dianne Prost O'Leary*

* Department of Mathematics, The University of Michigan, Ann Arbor, Michigan 48109

This work was supported in part by the Fannie and John Hertz Foundation, the National Science Foundation under Grant MCS-76-06595, the Energy and Research and Development Administration Contract EY-76-S-03-0326 PA #30, and the National Science Foundation Grant MCS75-13497.

ABSTRACT

In this paper we apply matrix splitting techniques and a conjugate gradient algorithm to the problem of minimizing a convex quadratic form subject to upper and lower bounds on the variables. This method exploits sparsity structure in the matrix of the quadratic form. Choices of the splitting operator are discussed and convergence results are established. We present the results of numerical experiments showing the effectiveness of the algorithm on free boundary problems for elliptic partial differential equations, and we give comparisons with other algorithms.

0. Introduction

The techniques developed in [4] will here be applied to a constrained optimization problem:

$$\min_{X} \; 1/2\, x^T A x - x^T b$$

$$c \leq x \leq d$$

where A is a symmetric $n \times n$ positive definite matrix. This quadratic programming problem often arises in a form such that the matrix A is large and has a nonrandom sparsity pattern. The applications considered here arise from the finite difference discretization of free boundary problems for elliptic partial differential equations. Problems of this form include models of water flow through a porous dam [2], the journal bearing [7], and torsion applied to a bar [3].

We describe in Section 1 a conjugate gradient algorithm due to Polyak [18] which is suitable for this problem and develop a modification which can exploit sparsity structure in the matrix A . In Section 2, we give alternatives for the scaling operator for the conjugate gradient iteration. First some matrix theory is developed for eigenvalues of submatrices, and then these results are used to establish bounds on the rates of convergence of the methods proposed. In Section 3 numerical experiments are presented

which explore the effectiveness of the conjugate gradient method with matrix splittings and compare it with other algorithms. In Section 4 we summarize our results.

We will use the following notational conventions. Capital letters will denote matrices, and small letters denote vectors or scalars. Components of vectors will be indexed by small letters as subscripts, while subvectors will have capital indices. Superscripts will denote iteration numbers.

1. Conjugate Gradient Algorithms for Quadratic Programming

The quadratic programming problem

$$\min_{x} \; 1/2\, x^T A x - x^T b \qquad (1)$$
$$c \leq x \leq d$$

with A an $n \times n$ symmetric and positive definite matrix and b, c, and d given n-vectors, often arises in the context of discretization of elliptic partial differential equations. A solution to this problem always exists, and it is necessarily unique.

An equivalent formulation of the quadratic programming problem can be established through the Kuhn-Tucker optimality conditions [See 13, Chapter 7]. For an arbitrary x ,

let y be defined by

$$y = Ax - b. \tag{2}$$

Then x solves (1) if and only if for $j = 1,2,\ldots,n$,

$$\begin{aligned} y_j &\geq 0 \quad \text{if} \quad x_j = c_j \\ y_j &\leq 0 \quad \text{if} \quad x_j = d_j \\ y_j &= 0 \quad \text{if} \quad c_j < x_j < d_j . \end{aligned}$$

An important special case of the quadratic programming problem is the linear complementarity problem, in which $c = 0$ and $d = \infty$. The optimality conditions then reduce to

$$\begin{aligned} x^T y &= 0 \quad \text{(complementarity condition)} \\ x \geq 0,\ y &\geq 0 \quad \text{(nonnegativity condition)} . \end{aligned}$$

The algorithm upon which we will build is an iterative method due to Polyak [18]. The Polyak algorithm maintains feasibility of the vector iterates $x^{(k)}$ (i.e., $c \leq x^{(k)} \leq d$) while iterating toward the proper sign conditions on y . Given an initial feasible $x^{(0)}$, the Polyak algorithm performs a series of nested iterations. In the outer iteration we choose a subset I of the indices $\{1,2,\ldots,n\}$ for which the variables x_i are at their upper or lower bounds and the optimality conditions are satisfied; specifically,

(3) $I = \{i: x_i = ci \text{ and } y_i > 0\} \cup \{i: xi = d_i \text{ and } y_i < 0\}$.

The vector of x variables whose indices belong to this set will be denoted x_I and all other x variables will be denoted by x_J . Corresponding to this choice of the index set I , we partition and rearrange the y and b vectors into y_I and y_J , and b_I and b_J respectively, and the matrix A is rearranged symmetrically. With this notation, (2) is equivalent to

$$\begin{pmatrix} A_{II} & A_{JI}^T \\ A_{JI} & A_{JJ} \end{pmatrix} \begin{pmatrix} x_I \\ x_J \end{pmatrix} - \begin{pmatrix} b_I \\ b_J \end{pmatrix} = \begin{pmatrix} y_I \\ y_J \end{pmatrix} .$$

The values of variables x_I will be kept fixed during the inner iteration, which will try to force all variables y_J to be zero by solving

$$(4) \qquad A_{JJ}x_J = b_J - A_{JI}x_I.$$

A_{JJ} is positive definite and symmetric because it is a principal submatrix of A , so the conjugate gradient method [14] can be applied to this linear system. We could solve this system exactly if we did not have upper and lower bounds on the variables, but because we want to keep these bounds satisfied, we modify the conjugate gradient iteration. If any step in the iteration would cause some

variable x_s with $s \in J$ to attain or to violate one of its bounds, the step is shortened if necessary to the point where x_s attains the bound, s is added to the set I (the index set of the unchanging variables), and the inner iteration is restarted with a new partitioning of the matrices and vectors. Once we complete the conjugate gradient iteration, we know that $y_J = 0$ and $c_J \leq x_J \leq d_J$, since the inner iteration solved (4) without violating any constraint on x_J. We then begin a new outer iteration, choosing, as in (3), an index set I corresponding to the current values of the variables x. If the new index set is the same as the one for the preceding cycle, then the optimality conditions are satisfied, and the algorithm halts with the solution. Otherwise a new inner iteration begins.

Now we will state the Polyak algorithm more precisely.

Initialization

- Choose an $x^{(0)}$ such that $c \leq x^{(0)} \leq d$, and set $k = 0$.

- Set $I = \{1,2,\ldots,n\}$. This definition ensures that the first halting test in the outer iteration will work properly.

Outer Iteration

Let $k = k + 1$, $x^{(k)} = x^{(k-1)}$, $y^{(k)} = Ax^{(k)} - b$, and $I_{k-1} = I$.

Define $I_k = \{i: x_i^{(k)} = c_i \text{ and } y_i^{(k)} > 0\} \cup \{i: x_i^{(k)} = d_i \text{ and } y_i^{(k)} < 0\}$.

If $I_k = I_{k-1}$, halt. The optimal solution has been found. Otherwise, set $I = I_k$ and begin the inner iteration.

Inner Iteration

(a) Partition and rearrange the matrix system as

$$x^{(k)} \to \begin{pmatrix} x_I^{(k)} \\ x_J^{(k)} \end{pmatrix}, \quad b \to \begin{pmatrix} b_I^{(k)} \\ b_J^{(k)} \end{pmatrix}, \quad A \to \begin{pmatrix} A_{II} & A_{JI}^T \\ A_{JI} & A_{JJ} \end{pmatrix}$$

with A_{JJ} $s \times s$, symmetric, and positive definite. We initialize the conjugate gradient iteration to solve equation (4). The sequence $\{z^{(q)}\}$ will be our approximations to the solution vector x_J. The vectors $p^{(q)}$ will be search directions, and vectors $r^{(q)}$ will be residuals for equation (4). Set $q = 0$ and

$$z^{(0)} = x_J^{(k)}$$

$$p^{(0)} = r^{(0)} = b_J - A_{JI}x_I^{(k)} - A_{JJ}z^{(0)}$$

(b) Calculate the new iterate and residual. We compute two step parameters: a_{cg} is the conjugate gradient step in the direction $p^{(q)}$, and a_{max} is the largest step in that direction which does not violate any bounds on the variables.

$$a_{cg} = \frac{(r^{(q)},p^{(q)})}{(p^{(q)},A_{JJ}p^{(q)})} = \frac{(r^{(q)},r^{(q)})}{(p^{(q)},A_{JJ}p^{(q)})}$$

$$a_{max} = \min\left(\min_{\substack{j=1,2,\ldots,s \\ p_j^{(q)} < 0}} \frac{c_j - z_j^{(q)}}{p_j^{(q)}} , \min_{\substack{j=1,2,\ldots,s \\ p_j^{(q)} > 0}} \frac{d_j - z_j^{(q)}}{p_j^{(q)}} \right)$$

The step taken is the smaller of these two positive numbers.

$$a_q = \min(a_{cg}, a_{max})$$

$$z^{(q+1)} = z^{(q)} + a_q p^{(q)}$$

$$r^{(q+1)} = r^{(q)} - a_q A_{JJ} p^{(q)}$$

The vector y could also be updated at this stage to correspond to the current values $x_I^{(k)}$ and $z^{(q+1)}$.

(c) Test for termination of the inner iteration:

If $r^{(q+1)} = 0$, set $x_J^{(k)} = z^{(q+1)}$ and restart the outer iteration.

If $\{j: z_j^{(q+1)} = c_j \text{ or } d_j\} = \phi$, proceed with (d).

Otherwise, set $x_J^{(k)} = z^{(q+1)}$ and $I = \{i: x_i^{(k)} = c_1 \text{ or } d_i\}$. If $I = \{1,2,\ldots,n\}$, then restart the outer iteration. Otherwise restart the inner iteration.

(d) Calculate the new search direction $p^{(q+1)}$, A_{JJ} conjugate to the old ones.

$$b_q = -\frac{(A_{JJ}p^{(q)}, r^{(q+1)})}{(p^{(q)}, A_{JJ}p^{(q)})} = \frac{(r^{(q+1)}, r^{(q+1)})}{(r^{(q)}, r^{(q)})}$$

$$p^{(q+1)} = r^{(q+1)} + b_q p^{(q)}$$

Replace q by q+1 and go to (b).

The initialization of $z^{(0)}, p^{(0)}, r^{(0)}$, and q in step (a) of the inner iteration, plus steps (b) and (d) with $a_q = a_{cg}$ and (c) replaced by

(c') If $r^{(q+1)} = 0$, then halt with $x_J = z^{(q+1)}$;

comprise the standard conjugate gradient algorithm for solving the linear system (4). The first iteration is equivalent to a steepest descent step for minimizing the quadratic form, and successive steps use as the search direction the component of the gradient which is A_{JJ} conjugate to all previous search directions.

The conjugate gradient method for solving positive definite linear systems terminates in a finite number of iterations. Moreover, $\{E(x^{(\ell)})\}$ is a monotonically decreasing sequence, where

$$E(x) = 1/2\,(x-x^*, A(x-x^*)) ,$$

x^* is the solution to the system $Ax^* = b$, and the iterates $x^{(\ell)}$ are obtained via the conjugate gradient algorithm [8]. We now show that the quadratic programming algorithm also has finite termination.

Theorem 1 Polyak's algorithm terminates in a finite number of iterations.

Proof: Each inner iteration terminates because either the chosen system is solved by conjugate gradients, or the size of the system is reduced (possibly several times) and the reduced system is solved by conjugate gradients. Let x'_J denote the solution to (4) for a particular choice of the set I and values x_I. We want to show that $E(x)$, the conjugate gradient descent function for solving $Ax^* = b$, is a descent function within the inner iteration. Now

$$E(x) = 1/2\ (x^T Ax - 2x^T b + x^{*T} b)$$

$$= 1/2\ (x_J^T A_{JJ} x_J + 2x_J^T A_{JI} x_I - 2x_J^T b_J)$$

$$+ 1/2\ (x_I^T A_{II} x_I - 2x_I^T b_I + x^{*T} b)$$

$$= 1/2\ (x_J - x_J', A_{JJ}(x_J - x_J'))$$

$$+ 1/2 (x_I^T A_{II} x_I - 2x_I^T b_I + x^{*T} b - x_J'^T A_{JJ} x_J')$$

The first term, $(x_J - x_J', A_{JJ}(x_J - x_J'))/2$, is the conjugate gradient descent function for solving the linear system (4), and the rest of the expression for $E(x)$ is constant within the inner iteration, so $E(x)$ has been shown to be a descent function for any inner iteration between restarts. But any restart of the conjugate gradient -algorithm will preserve the descent property, so $E(x)$ is a descent function for the entire algorithm. Thus no linear system can repeat once it has been solved in an inner iteration, and since there are finitely many linear systems (corresponding to a choice of index set and the choice of either upper or lower bound for each variable in it), the algorithm must terminate. ∎

Diamond's algorithm [10] is a special case of Polyak's for problems with $c = 0, d = \infty$ and A an M-matrix. In that case, the chosen system for the inner iteration can always be solved without violating the constraints on x_J, and

it can be shown that the subsets I are nested:

$$I_{k+1} \subset I_k .$$

Diamond chooses to solve the linear problems in the inner iteration by an iterative method other than conjugate gradients.

The performance of the Polyak or the Diamond algorithm can be greatly enhanced by improving the convergence rate of the inner iterations. This can be accomplished by using the scaled conjugate gradient algorithm with matrix splittings described in [4]. In this algorithm, we base our search direction p on $\bar{M}^{-1}r$ rather than on r, where $\bar{M}^{-1}$ is an approximation to the matrix A_{JJ}^{-1}. One precaution must be taken, however. A problem may arise if, in beginning the inner iteration, some x_s is at its bound for $s \in J$. Suppose, for example, that $x_s = c_s$ and $r_s > 0$. (A negative value for r_s would imply that $s \in I$.) Then for the normal conjugate gradient iteration, $p^{(0)} = r^{(0)}$, so $p_s^{(0)} > 0$ and the step increases x_s since the step parameter a_0 is positive. Thus the bound on x_s remains satisfied. If we apply the scaled algorithm, however, $(\bar{M}^{-1} r^{(0)})_s$ may be negative and the algorithm would not be able to take a step without violating the constraint that $x_s \geq c_s$. We avoid this problem by performing one initial steepest descent step $(p^{(0)} = r^{(0)})$ at the beginning of each

-inner iteration and then proceeding with the scaled algorithm.

The resulting algorithm is as follows

Initialization

- Choose an $x^{(0)}$ such that $c \leq x^{(0)} \leq d$, and set $k = 0$.

Set $I = \{1,2,...,n\}$.

Outer Iteration

- Let $k = k+1$, $x^{(k)} = x^{(k-1)}$, $y^{(k)} = Ax^{(k)}-b$, and $I_{k-1} = I$.

- Define $I_k = \{i: x_i^{(k)} = c_1$ and $y_i^{(k)} > 0\} \cup \{i: x_i^{(k)} = d_i$ and $y_i^{(k)} < 0\}$.

If $I_k = I_{k-1}$, halt. The optimal solution has been found. Otherwise, Set I = Ik and begin the inner iteration.

Inner Iteration

(a) Partition and rearrange the matrix system as

$$x^{(k)} \to \begin{pmatrix} x_I^{(k)} \\ x_J^{(x)} \end{pmatrix}, \; b \to \begin{pmatrix} b_I^{(k)} \\ b_J^{(k)} \end{pmatrix}, \; A \to \begin{pmatrix} A_{II} & A_{JI}^T \\ A_{JI} & A_{JJ} \end{pmatrix}$$

with A_{JJ} $s \times s$, symmetric, and positive definite.
We initialize the iteration to solve equation (4). Set

$$z^{(0)} = x_J^{(k)}$$

$$r^{(0)} = b_J - A_{JI}x_I^{(k)} - A_{JJ}z^{(0)} .$$

(b) Calculate the new iterate and residual. We calculate two step parameters: a_{cg} is the conjugate gradient, or, equivalently for this step, the steepest descent parameter, and a_{max} is the largest step which does not violate any of the bounds.

$$a_{cg} = \frac{(r^{(0)}, r^{(0)})}{(r^{(0)}, A_{JJ}r^{(0)})}$$

$$a_{max} = \min\left(\min_{\substack{j=1,2,\ldots,s \\ r_j^{(0)} < 0}} \frac{c_j - z_j^{(0)}}{r_j^{(0)}} , \min_{\substack{j=1,2,\ldots,s \\ r_j^{(0)} > 0}} \frac{d_j - z_j^{(0)}}{r_j^{(0)}} \right)$$

The step taken is the smaller of these two positive numbers.

$$a_0 = \min(a_{cg}, a_{max})$$

$$z^{(1)} = z^{(0)} + a_0 r^{(0)}$$

$$r^{(1)} = r^{(0)} - a_0 A_{JJ} r^{(0)}$$

If $r^{(1)} = 0$, set $x_J^{(k)} = z^{(1)}$ and restart the outer iteration.

If $\{j: z_j^{(1)} = c_j \text{ or } d_j\} = \phi$, proceed with (c).

Otherwise, set $x_J^{(k)} = z^{(1)}$ and $I = \{i: x_i^{(k)} = c_i \text{ or } d_i\}$. If $I = \{1,2,\ldots,n\}$, then restart the outer iteration. Otherwise repartition x, b, and A as in (a), set

$$z^{(1)} = x_J^{(k)}$$

$$r^{(1)} = b_J - A_{JI}\, x_I^{(k)} - z^{(1)} ,$$

and continue with (c).

(c) Initialize the scaled conjugate gradient algorithm. Choose $\bar{M}$ to scale the matrix A_{JJ} , set $q = 1$, and

$$p^{(1)} = \bar{M}^{-1} r^{(1)} .$$

(d) Calculate the new iterate and residual:

$$a_{cg} = \frac{(r^{(q)}, p^{(q)})}{(p^{(q)}, A_{JJ} p^{(q)})} = \frac{(r^{(q)}, \bar{M}^{-1} r^{(q)})}{(p^{(q)}, A_{JJ} p^{(q)})}$$

$$a_{max} = \min\left(\min_{\substack{j=1,2,\ldots,s \\ p_j^{(q)} < 0}} \frac{c_j - z_j^{(q)}}{p_j^{(q)}} , \min_{\substack{j=1,2,\ldots,s \\ p_j^{(q)} > 0}} \frac{d_j - z_j^{(q)}}{p_j^{(q)}} \right)$$

$$a_q = \min(a_{cg}, a_{max})$$

$$z^{(q+1)} = z^{(q)} + a_q p^{(q)}$$

$$r^{(q+1)} = r^{(q)} - a_q A_{JJ} p^{(q)}$$

(e) Test for termination of the inner iteration:

If $r^{(q+1)} = 0$, set $x_J^{(k)} = z^{(q+1)}$ and restart the outer iteration.

If $\{j: z_j^{(q+1)} = c_j \text{ or } d_j\} = \phi$, proceed with (f).

Otherwise, set $x_J^{(k)} = z^{(q+1)}$ and $I = \{i: x_i^{(k)} = c_i \text{ or } d_i\}$. If $I = \{1,2,\ldots,n\}$ then restart the outer iteration. Otherwise restart the inner iteration.

(f) Calculate the new search direction, A_{JJ} orthogonal to the old ones.

$$b_q = - \frac{(A_{JJ} p^{(q)}, \bar{M}^{-1} r^{(q)})}{(p^{(q)}, A_{JJ} p^{(q)})} = \frac{(r^{(q+1)}, \bar{M}^{-1} r^{(q+1)})}{(r^{(q)}, \bar{M}^{-1} r^{(q)})}$$

$$p^{(q+1)} = \bar{M}^{-1} r^{(q+1)} + b_q p^{(q)}$$

Replace q by q+1 and go to (d).

Initialization of $z^{(1)}$, $r^{(1)}$ and q, plus steps (c), (d), and (f) with $a_q = a_{cg}$ and (e) replaced by

(e') If $r^{(q+1)} = 0$ then halt with $x_J = z^{(q+1)}$

comprise the scaled conjugate gradient algorithm for solving the linear system (4). [See 4].

Since E(x) is a descent function for both the original conjugate gradient algorithm and the scaled version [4], the convergence proof given above for Polyak's algorithm applies to the modified version, too.

One further refinement is possible in the computation. We do not need to solve the linear systems in the inner iteration to a high level of accuracy, since the sole purpose of this step is to determine the next index set I we wish to consider. We need only guarantee that no system will repeat. Thus we can work with a large error tolerance and test whether $\|r^{(q+1)}\| < \varepsilon_k$ in step (e), rather then whether $r^{(q+1)} = 0$. This tolerance is refined before termination in the solution of the final linear system. This device reduced the number of operations in the computation by a factor close to two in numerical experiments.

Thus far we have developed a finite algorithm to solve the quadratic programming problem with upper and lower bounds. The algorithm never changes the matrix A and in fact only needs to use A to form products with arbitrary vectors. Thus the algorithm is suitable for sparse matrices A .

2. The Choice of the Scaling Matrix $\bar{M}$

A remaining issue is the choice of the matrix $\bar{M}$. We need a scaling matrix $\bar{M}$ such that the computation of $\bar{M}^{-1}r$ can be performed easily and so that the convergence of the conjugate gradient algorithm is accelerated significantly. The convergence rate for the conjugate gradient method applied to the linear system is bounded as follows:

$$E(x^{(k)}) \leq (1-\kappa^{-1})\, E(x^{(k-1)}) \tag{5}$$

where κ is the ratio of the largest and smallest eigenvalues of the matrix $\bar{M}^{-1/2} A_{JJ} \bar{M}^{-1/2}$ and E is the descent function for equation (4). [8]

We consider in this section two classes of scaling matrices. The first class is determined by the knowledge of good scaling matrices for the full operator A , and the second class is formed by applying alternate iterative methods to the quadratic programming problem.

2.1 Methods Based on a Scaling of the Matrix A

Suppose that M is a positive definite scaling matrix for A and that P is the permutation matrix corresponding to the current partitioning and rearrangement of the linear system:

$$PAP^T = \begin{pmatrix} A_{II} & A_{JI}^T \\ A_{JI} & A_{JJ} \end{pmatrix}.$$

There are three simple methods which could be used to obtain a matrix $\bar{M}$ whichscales A_{JJ} .

Method 1:

Partition and rearrange the matrix M in a manner corresponding to the current rearrangement of A

$$PMP^T = \begin{pmatrix} M_{II} & M_{JI}^T \\ M_{JI} & M_{JJ} \end{pmatrix}$$

and use M_{JJ} as the scaling matrix $\bar{M}$.

Method 2:

Partition and rearrange the matrix $W = M^{-1}$ in a manner corresponding to the current rearrangement of A :

$$PWP^T = \begin{pmatrix} W_{II} & W_{JI}^T \\ W_{JI} & W_{JJ} \end{pmatrix}$$

and use W_{JJ}^{-1} as the scaling matrix $\bar{M}$.

Method 3:

If a Cholesky factorization of M is available, partition and rearrange the factors LL^T as

$$PLL^TP^T = (PLP^T)(PL^TP^T) = \begin{pmatrix} L_{II} & L_{IJ} \\ L_{JI} & L_{JJ} \end{pmatrix} \begin{pmatrix} L_{II}^T & L_{JI}^I \\ L_{IJ}^I & L_{JJ}^T \end{pmatrix}$$

and use $L_{JJ}L_{JJ}^I$ as $\bar{M}$.

In actual computation, the matrices and vectors are never physically rearranged. A vector of logical variables can indicate membership in I or J and can be used to ignore the appropriate matrix or vector elements.

In special cases a single factorization of $M = LL^T$, where L is lower triangular, suffices for Method 1. Consider a tridiagonal matrix of the form

$$M = \begin{pmatrix} M_1 & & & \\ & M_2 & & \\ & & \ddots & \\ & & & M_t \end{pmatrix}_{n \times n}$$

where

$$M_i = \begin{pmatrix} m_1 & m_2 & & & \\ m_2 & m_1 & m_2 & & \\ & \ddots & \ddots & \ddots & \\ & & m_2 & m_1 & \\ & & & & \end{pmatrix}_{\alpha_i \times \alpha_i}, \quad \alpha_1+\alpha_2+\ldots+\alpha_t = n$$

Then $\bar{M}$ has the form

$$\begin{pmatrix} \bar{M}_1 & & & \\ & \bar{M}_2 & & \\ & & \ddots & \\ & & & \bar{M}_\mu \end{pmatrix}_{s \times s}, \quad \bar{M}_i : \beta_i \times \beta_i, \quad \beta_1+\beta_2+\ldots+\beta_\mu = s ,$$

where. $\bar{M}_i$ has the same form as the matrix M_i , but different dimension. So the factors of each block $\bar{M}_i$ are the leading principal submatrices of the factors L and L^T of the largest matrix M_i in M .

Although Method 2 seems to be the most complicated, it can easily be implemented without forming M^{-1} . Since $W_{JJ} = (M_{JJ}-M_{JI}M_{II}^{-1}M_{JI}^T)^{-1}$, we can form $y2 = W_{JJ}r$ by solving the system

$$\begin{pmatrix} M_{II} & M_{JI}^T \\ M_{JI} & M_{JJ} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} 0 \\ r \end{pmatrix}$$

Thus it suffices to have a subroutine to set up the right hand side, solve a linear system with the original matrix M, and pick the appropriate elements from the solution vector y. The disadvantage of this technique is that it is much slower than the others if the set I has many elements, since we must work with a full size matrix system each time.

We now wish to show that whenever $\bar{M}$ is obtained from a matrix M by one of the three methods above, then the convergence bound for the conjugate gradient method applied to a linear system involving the matrix A_{JJ} using the scaling matrix $\bar{M}$ is at least as good as that for the conjugate gradient method applied to a linear system involving the full matrix A with scaling M. To do this, we compare the eigenvalues of $\bar{M}^{-1}A_{JJ}$ with those of $M^{-1}A$ and thus get a bound on κ in expression (5). For any positive definite scaling matrix M we have the following results:

Lemma 1 Let the scaling matrix M be obtained using Method 1 or Method 2 above. Then it is positive definite. Suppose the dimension of $\bar{M}$ is $n-l$, and let

$\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n > 0$ berootsof $\det(A-\lambda M) = 0$ and

$\bar{\lambda}_1 \geq \bar{\lambda}_2 \geq \ldots > \bar{\lambda}_{n-1} > 0$ be roots of $\det(A_{JJ}-\lambda\bar{M}) = 0$.

Then $\lambda_1 \geq \bar{\lambda}_1 \geq \lambda_2 \geq \bar{\lambda}_2 \geq \ldots \geq \lambda_{n-1} \geq \bar{\lambda}_{n-1} \geq \lambda_n$.

Proof: $\bar{M}$ is positive definite since it is a principal submatrix of a positive definite matrix. For the proof of the interlacing of the eigenvalues, see Wilkinson [20, p.340] ∎

Lemma 2 Let the scaling matrix $\bar{M}$ be obtained using Method 3 above. Then the results of Lemma 1 hold for it.

Proof: The main diagonal elements of the factor L_{JJ} are a subset of the main diagonalelementsof L , which are all non-zero since LL^T is positive definite. Thus $L_{JJ}L_{JJ}^T$ is positive definite, too. To prove that the eigenvalues interlace, note that

$$\det(A-XM) = \det(A-\lambda LL^T) = \det(L^{-1}AL^{-T}-\lambda I) .$$

By the Courant-Fischer characterization of eigenvalues,

$$\lambda_{\sigma+1} = \min_{P_{\sigma\times n}} \max_{x}\{x^T L^{-1}AL^{-T}x: \quad \|x\| = 1,\ Px = 0\}$$

$$= \min_{P_{\sigma\times n}} \max_{Y}\{ y^T Ay: \quad \|L^T y\| = 1,\ Py = 0\}, \sigma = 0,1,\ldots,n-1$$

where P is any matrix of the indicated dimension.

Suppose that A_{JJ} is obtained from A by deleting the k-th row and column. Then

$$\bar{\lambda}_{\sigma+1} = \min_{P_{\sigma\times n-1}} \max_{x_J}\{x_J^T L_{JJ}^{-1} A_{JJ} L_{JJ}^{-T} x_J: \quad \|x_J\| = 1,\ Px_J = 0\}$$

$$= \min_{P_{\sigma\times n-1}} \max_{y_J}\{y_J^T A_{JJ} y_J: \quad \|L_{JJ}^T y_J\| = 1,\ Py_J = 0\}$$

$$= \min_{P_{\sigma\times n}} \max_{Y}\{y^T Ay: \quad yk = 0, (L^T y)k = 0,\ \|L^T y\| = 1, Py = 0\}$$

$$= \min_{P_{\sigma\times n}} \max_{Y}\{y^T Ay: \|L^T y\| = 1,\ Py = 0,\ e_k^T y = 0, e_k^T L^T y = 03$$

where ek is the k-th unit vector. Therefore, $\bar{\lambda}_{\sigma+1} \leq \lambda_{\sigma+1}$.

Similarly,

$$\lambda_{\sigma+1} = \max_{P_{n\text{-}a\text{-}1\times n}} \min_{X}\{x^T L^{-1} A L^{-T} x: \quad \|X\| = 1,\ Px = 0\}$$

$$= \max_{P_{n\text{-}a\text{-}1\times n}} \min_{Y}\{y^T Ay: \|L^T y\| = 1,\ Py = 0\}, \sigma = 0,1,\ldots,n-1$$

$$\bar{\lambda}_{\sigma} = \max_{P_{n\text{-}a\text{-}1\times n-1}} \min_{x_J}\{x_J^T L_{JJ}^{-1} A_{JJ} L_{JJ}^{-T} x_J: \|x_J\| = 1, Px_J = 0\}$$

$$= \max_{P_{n-\sigma-1\times n}} \min_{Y}\{y^T Ay: \|L^T y\| = 1,\ Py = 0, e_k^T y = 0, e_k^T L y^T = 0\}$$

Therefore, $\bar{\lambda}_{\sigma} > \lambda_{\sigma+1}$ and the result follows. ∎

Lemma 3 If $\bar{M}$ is obtained by either Method 1, Method 2, or Method 3, then if λ_1 and λ_n are respectively the largest and smallest roots of $\det(A-AM) = 0$, and $\bar{\lambda}_1$ and $\bar{\lambda}_s$ are respectively the largest and smallest roots of $\det(A_{JJ}-\lambda\bar{M}) = 0$, where the matrices $\bar{M}$ and A_{JJ} have dimension s, then $\lambda_1 \geq \bar{\lambda}_1$ and $\lambda_n \leq \bar{\lambda}_s$.

Proof: This result follows from induction using the results of Lemmas 1 and 2. ∎

Lemma 3 gives us the following result:

Theorem 2 The convergence bound for the conjugate gradient algorithm applied to the subproblems is at least as good as that of the conjugate gradient method applied to the original matrix.

Thus, if we have a matrix M for which linear systems $M d = r$ can be solved easily, and M scales A well in the sense that the roots of $\det(A-\lambda M)$ do not have a wide range, then we have a good scaling operator for the subproblems in the scaled conjugate gradient algorithm for quadratic programming.

The simplest scaling matrix M is the diagonal portion of A $(m_{\ell\ell} = a_{\ell\ell}, m_{\ell j} = 0\ \ell,j = 1,2,\ldots,n,\ell\neq j)$. It has been shown by Forsythe and Straus [12] that if A is two-cyclic, then among all diagonal matrices, this choice

minimizes κ in (5) and thus maximizes the estimated convergence rate. Even for a general matrix A , it is often advantageous to scale the problem in this way.

From the form of the matrix M in Method 3 , we can see that the matrices $\bar{M}$ for Methods 1 and 3 differ by at most a rank n-s matrix, where s is the dimension of $\bar{M}$, and the eigenvalues of the matrix obtained by Method 1 are greater than or equal to the eigenvalues of the matrix obtained by Method 3 .

2.2 Methods Based on Iterative Algorithms

It has been shown before [For example, 1] that suitable iterative techniques for solving linear or nonlinear systems can be accelerated by application of the conjugate gradient algorithm. We can extend this idea to our problem. Define $\bar{M}^{-1} r^{(i)}$ by $z^{(i)} - \bar{z}$ where $\bar{z}$ is the vector obtained by applying a double sweep of modified symmetric successive over-relaxation (SSOR) to the linear system (4) using $z^{(i)}$ as the initial guess. The SSOR iteration is modified so that no variable violates the constraints. More precisely, let

$$f_J = b_J - A_{JI} x_I$$

$$A_{JJ} = (\alpha_{j\ell})_{s \times s}$$

We apply the SSOR iteration to the system

$$A_{JJ}z = f_J$$

$$c_J \leq z \leq d_J$$

For $j = 1,2,\ldots,s$, let

$$z_j^f = z_j^{(i)} + \omega\Big(f_j - \sum_{\ell=1}^{j-1} a_{j\ell}\tilde{z}_\ell - \sum_{\ell=j}^{S} a_{j\ell} z_\ell^{(i)}\Big)/\alpha_{jj}$$

$$\tilde{z}_j = \begin{cases} c_j & \text{if } z_j^f < c_j \\ d_j & \text{if } z_j^f > d_j \\ z_j^f & \text{otherwise} \end{cases}$$

and for $j = s,s-1,\ldots,1$, let

$$z_j^b = \tilde{z}_j + \omega\Big(f_j - \sum_{\ell=1}^{j} \alpha_{j\ell}\tilde{z}_\ell - \sum_{\ell=j+1}^{S} \alpha_{j\ell}\bar{z}_\ell\Big)/\alpha_{jj}$$

$$\bar{z}_j = \begin{cases} c_j & \text{if } z_j^b < c_j \\ d_j & \text{if } z_j^b > d_j \\ z_j^b & \text{otherwise} \end{cases}$$

where ω is a parameter such that $0 < \omega < 2$. Then the result of one iteration of modified SSOR is $\bar{z}$. The nonsymmetric version of this iteration (using forward sweeps only) has been discussed by Cottle and Goheen [5]. for problems with A an M-matrix.

For the modified SSOR iteration, the scaling operator $\bar{M}^{-1}$ has no simple form. The matrix is neither symmetric nor positive definite, and it changes from iteration to iteration in the conjugate gradient algorithm. Thus, none of the conjugate gradient convergence theory applies. No,netheless, it has performed well in experiments on elliptic partial differential equations.

As mentioned in Section 1, for the special case in which $C = 0$, $d = \infty$ and A is an M-matrix, the linear systems can always be solved without violating the constraints on x_J. In this case, we can simply set

$$\tilde{z}_j = z_j^f \quad \text{and} \quad \bar{z}_j = z_j^b$$

without degrading the convergence of the iteration, reducing the matrix $\bar{M}^{-1}$ to

$$\bar{M}^{-1} = \omega(2-\omega)(I-\omega L^T)^{-1}(I-\omega L)^{-1}D^{-1}$$

where $A_{JJ} = D(I-L-L^T)$, L is strictly lower triangular, and D is diagonal. As long as A_{JJ} is normalized so that its diagonal elements are equal, this matrix is symmetric and positive definite, and the conjugate gradient convergence theory applies.

3. Alternate Algorithms and Numerical Results

Standard algorithms for the general quadratic programming problem involve complementary pivoting and inversion or factorization of submatrices of A [9,11,13,15,17]. These algorithms may not be practical for large, sparse, structured matrices. For example, free boundary problems in elliptic partial differential equations often give rise to irreducible Minkowski matrices (M-matrices), and A^{-1} may be totally full even though A is highly sparse. Successful algorithms for this special application of quadratic programming have often involved some modification of the SOR algorithm. Cea and Glowinski [3] propose a block form of the modified SOR iteration discussed in Section 2.2. Cryer [7] obtained good results with the specialization of this algorithm to the linear complementarity problem. Cottle, Golub, and Sacher [6] propose a SOR algorithm for the complementarity problem which uses Sacher's algorithm [19] for subproblems involving linear complementarity problems with tridiagonal matrices. Cottle and Goheen [5] extend this algorithm to the quadratic programming problem and survey several alternate methods.

We now present a summary of the results of numerical experiments on three groups of problems. We compare the

performance of the algorithm proposed in this paper with that of Cottle and Goheen's SOR algorithm discussed in Section 2.2, since in experiments reported in [5], it ranked among the most effective algorithms.

Example 1 The first problem is the linear complementarity problemwiththe matrix A corresponding to the Laplacian 5-point finite difference operator:

$$A = \begin{pmatrix} T & -I & & & \\ -I & T & -I & & \\ & \ddots & \ddots & \ddots & \\ & & -I & T & -I \\ & & & -I & T \end{pmatrix}_{m^2 \times m^2}, \quad T = \begin{pmatrix} 4 & -1 & & & \\ -1 & 4 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 4 & -1 \\ & & & -1 & 4 \end{pmatrix}_{m \times m}$$

The conjugate gradient algorithm was run with scaling matrices equal to the tridiagonal portion of A , a partial LL^T factorization, and the SSOR operator. (These algorithms are denoted in the tables and figures by CG + T, CG + LL^T , and CG + SSOR respectively), The LL^T factorization was chosen to be one for which L has the same sparsity patternas the lower triangular portion of A . The algorithm is due to Meijerink and van der Vorst and is defined in [16]. The scaling was performed using Methods 2 and 3 for the tridiagonal and LL^T matrices, but there was no significant difference between the

performance of the two methods. The SSOR scaling was also performed in each of the two ways discussed in Section 2.2, and, as expected, there was no difference in performance for this example problem. Table 1 shows the results of numerical experiments with randomly generated vectors b. We present the average number of inner iterations over five examples for the various algorithms and for $m = 16$ and $m = 23$ ($n = 256$ and 529 variables respectively). For the algorithms with parameter ω, results shown are the average over $\omega = 1.1, 1.3, 1.5, 1.7$, and 1.9. In all cases, the initial guess $x^{(0)}$ was 0, and $\varepsilon = 10^{-3}$ for all but the last iterations, with a final criteria of $\varepsilon = 10^{-6}$.

The conjugate gradient algorithms required 5-7 outer iterations for $n = 256$ and 6-8 for $n = 529$, independent of scaling. The average number of active variables per outer iteration was $s = 196$ for $n = 256$ and $s = 435$ for $n = 529$.

There is, of course, a varying amount of work per iteration depending on which scaling is used. The tridiagonal scaling from Method 3, for example, requires approximately $3s$ operations (multiplications and additions) while SSOR requires the equivalent of two matrix-vector multiplications involving the matrix A_{JJ} ($s \times s$). The SOR algorithm requires a matrix-vector

TABLE 1 Number of Iterations fdr Example 1

	CG with Tridiagonal Scaling Method 3 (CG+T)	CG with Partial LL^T Scaling Method 3 (CG + LL^T)	CG with SSOR Scaling (CG+SSOR)	SOR Algorithm
n=256	67	35	38	94
n=529	67	60	58	> 212

TABLE 2 Average Number of Variables Not at Their Bounds During the Conjugate Gradient Iteration for Example 2

n	s			s/n		
	c = 5	c = 9	c = 13	c = 5	c = 9	c = 13
256	185	138	109	.72	.54	.43
529	399	277	234	.75	.52	.44
900	662	473	393	.74	.53	.44

multiplication by the entire matrix A (n×n) at every -iteration, regardless of how many variables are at their bounds.

It can be shown that κ for the matrix A and for the matrix $M^{-1}A$ with tridiagonal scaling is $O(m^2)$. Using the optimal value of ω, SOR is expected to converge in $O(m^2)$ iterations when applied to a linear system involving the matrix A. The number of iterations for the quadratic programming algorithm is predicted well by the linear theory.

Figure 1 shows the variation in average number of iterations for different values of the parameter ω in the SOR algorithm and for conjugate gradients with SSOR scaling. The conjugate gradient algorithm can be seen to be much less sensitive to the choice of ω.

Example 2 This is a model for studying the effects of torsion applied to a rectangular bar. Cea and Glowinski [3] present the model for a crossection of the bar as follows

$$\min_{u} \; 1/2 \iint_{\Omega} |\nabla u|^2 dxdy - C \iint_{\Omega} u \; dx \; dy$$

$$u = 0 \quad \text{on} \quad \Gamma$$

$$|u(x,y)| \leq D(x,y,\Gamma)$$

where C is a positive constant related to the magnitude of the torsion, $D(x,y,\Gamma)$ is the distance between the

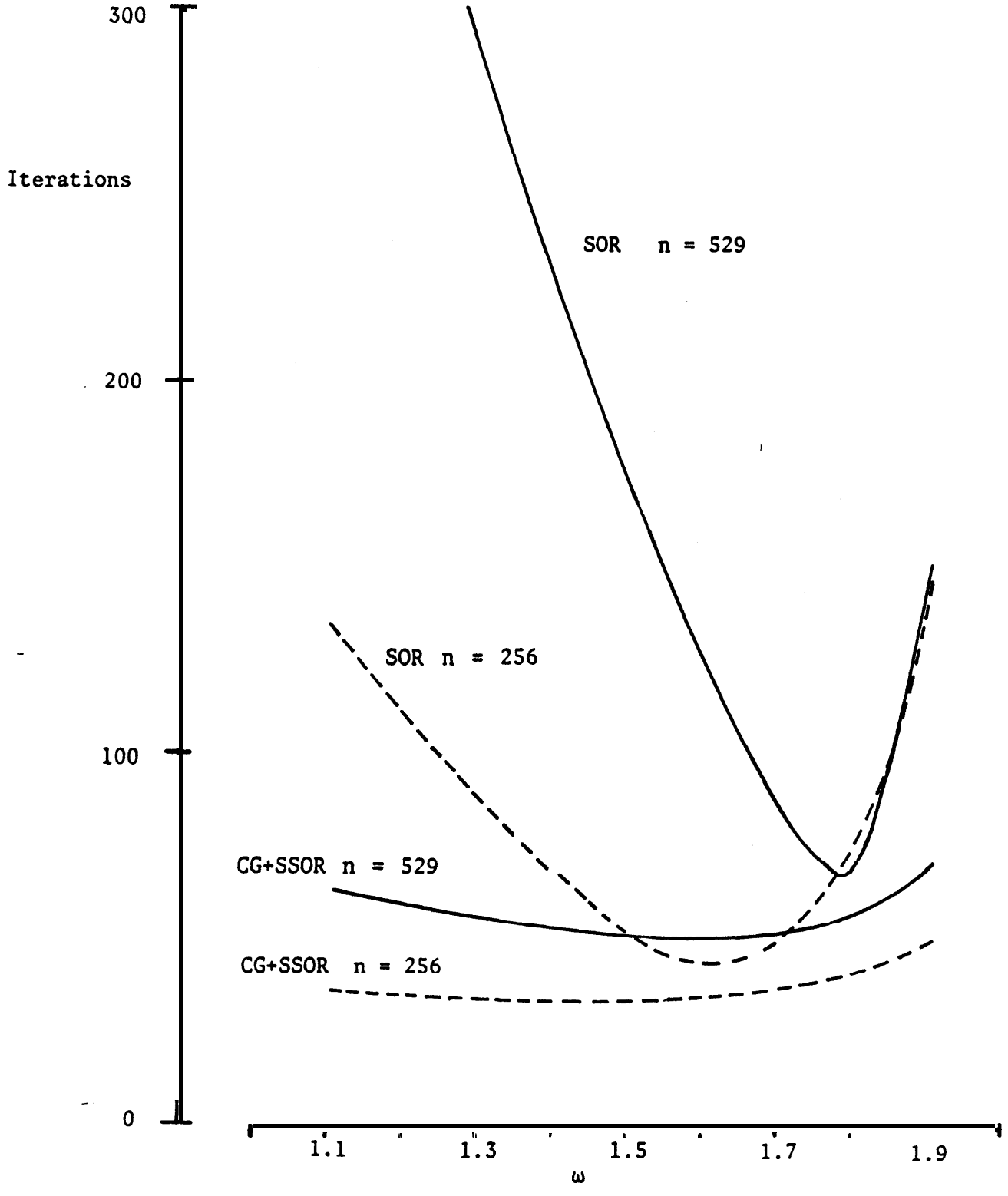


Figure 1. Algorithm Performance on Example 1 with Varying ω .

point (x,y) and Γ, the boundary of the region Ω, and u is the stress function. After discretization, this is a quadratic programming problem. The distances form the upper and lower bounds, the matrix A is taken to be the Laplacian 5-point operator, and b has every component equal to C. Figures 2-4 show the results of experiments with m = 16, 23, and 30 (n = 256, 529, and 900 respectively) and C = 5, 9, and 13. The initial guess and the convergence tolerance were as in Example 1. Increasing values of C correspond to more variables at their bounds in the final solution (approximately 30% for C = 5, 60% for C = 9, and 80% for C = 13). The constraints for this problem are much tighter than those for Example 1, and the second SSOR scaling for conjugate gradients is not effective here.

Figures 5-7 show the variation in convergence for various values of ω for the SOR algorithm and the conjugate gradient algorithm with SSOR scaling. Results are similar to those of Example 1, but in this problem, where so many variables are at their bounds in the optimal solution, it is even more important to take advantage of the reduction in work achieved by partitioning the system instead of working with the entire set of variables at each iteration. The average number of active variables is given in Table 2, and the number of outer iterations varied from 4 to 8 for n = 256, and from 5 to 11 for n = 900.

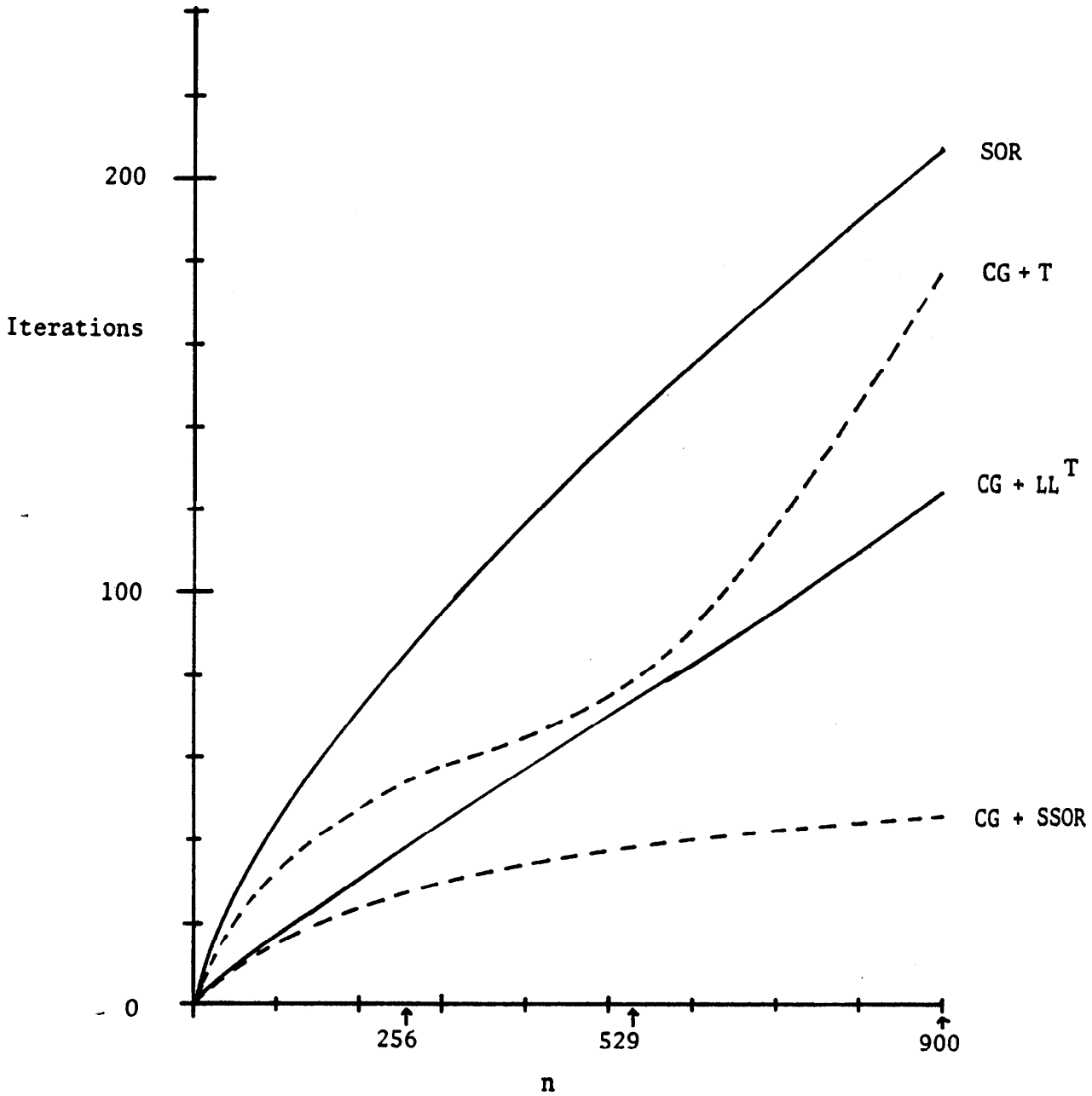


Figure 2. Algorithm Performance on Example 2, C = 5 .

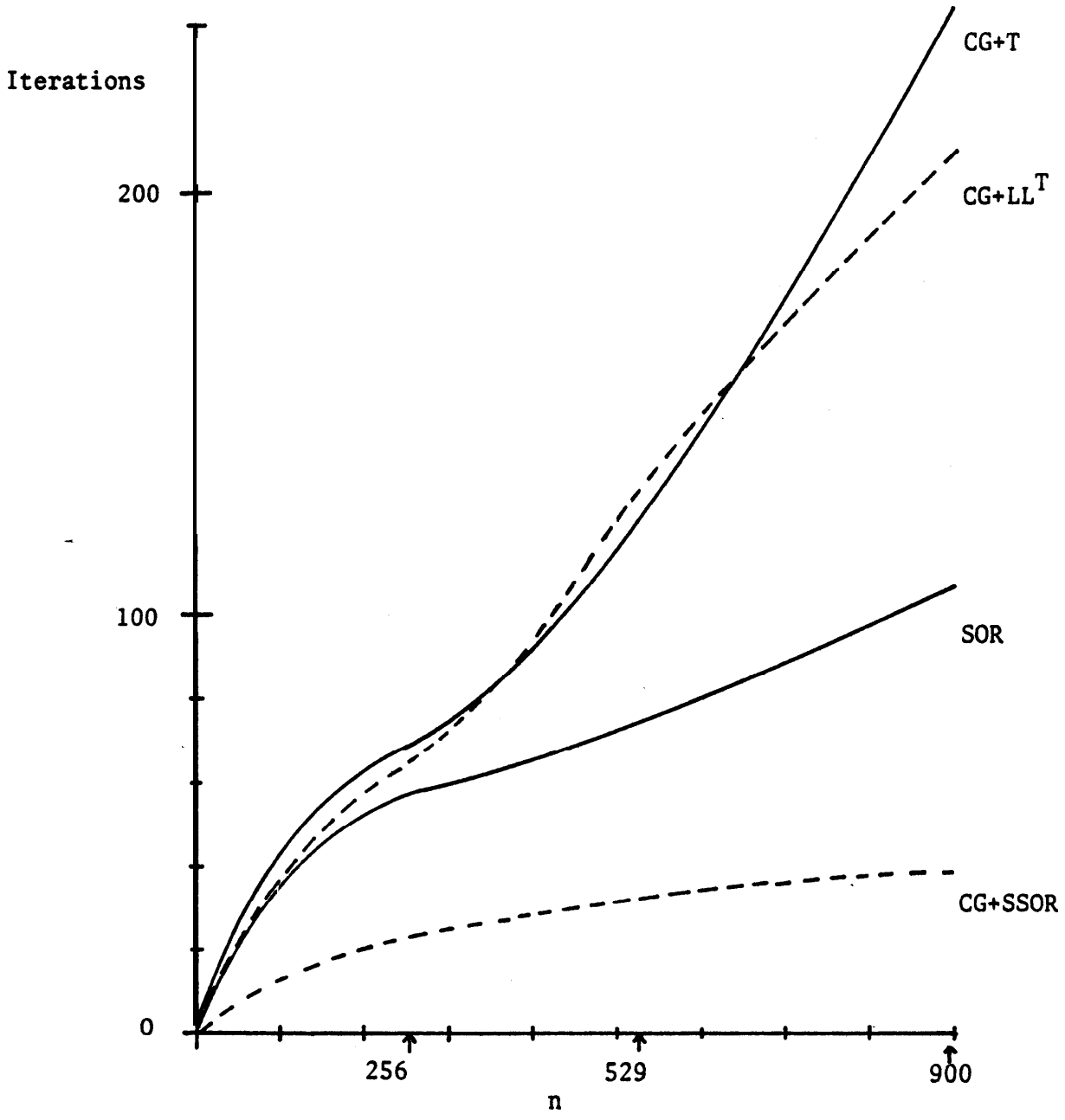


Figure 3. Algorithm Performance on Example 2, C = 9.

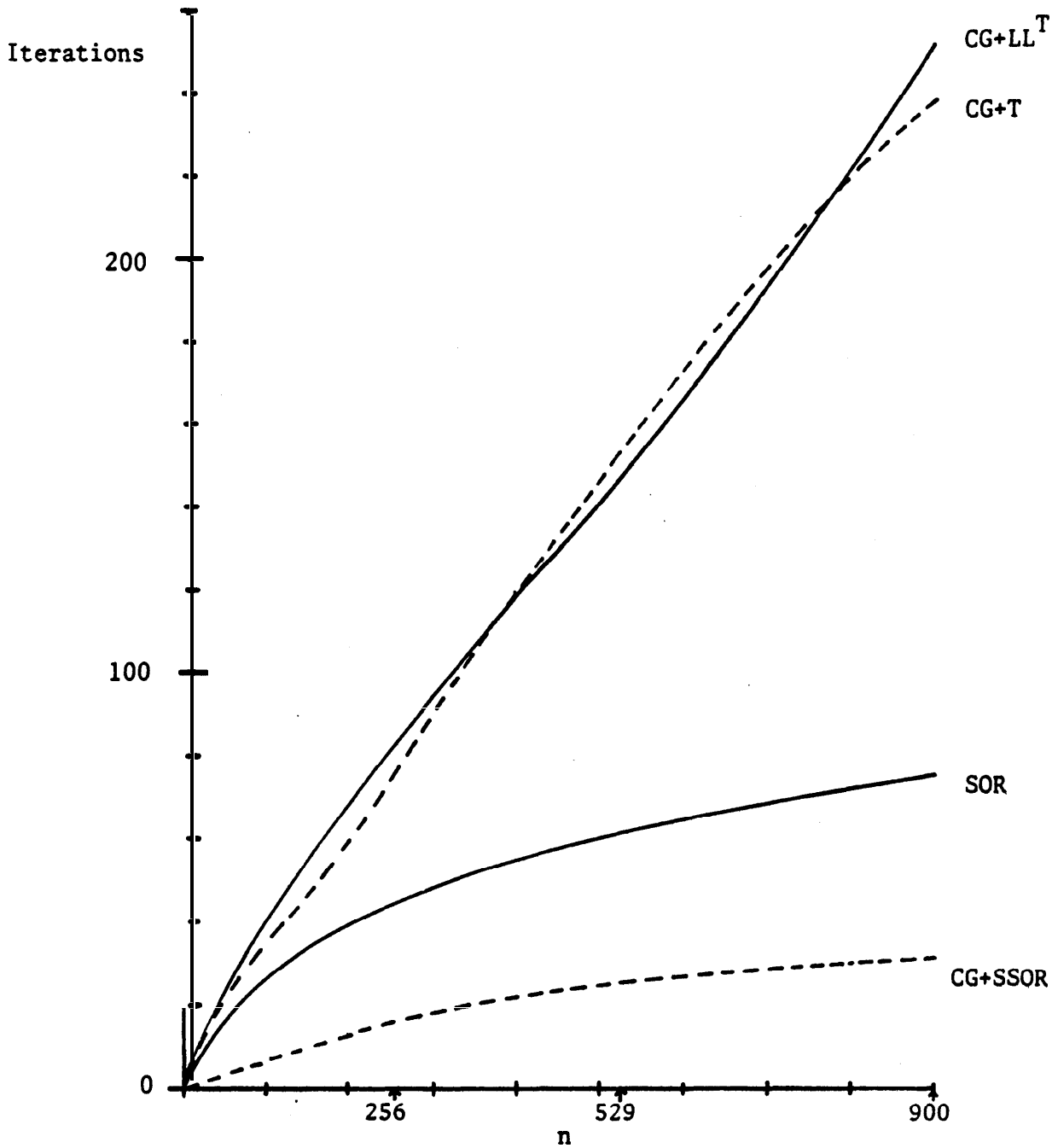


Figure 4. Algorithm Performance on Example 2, C = 13.

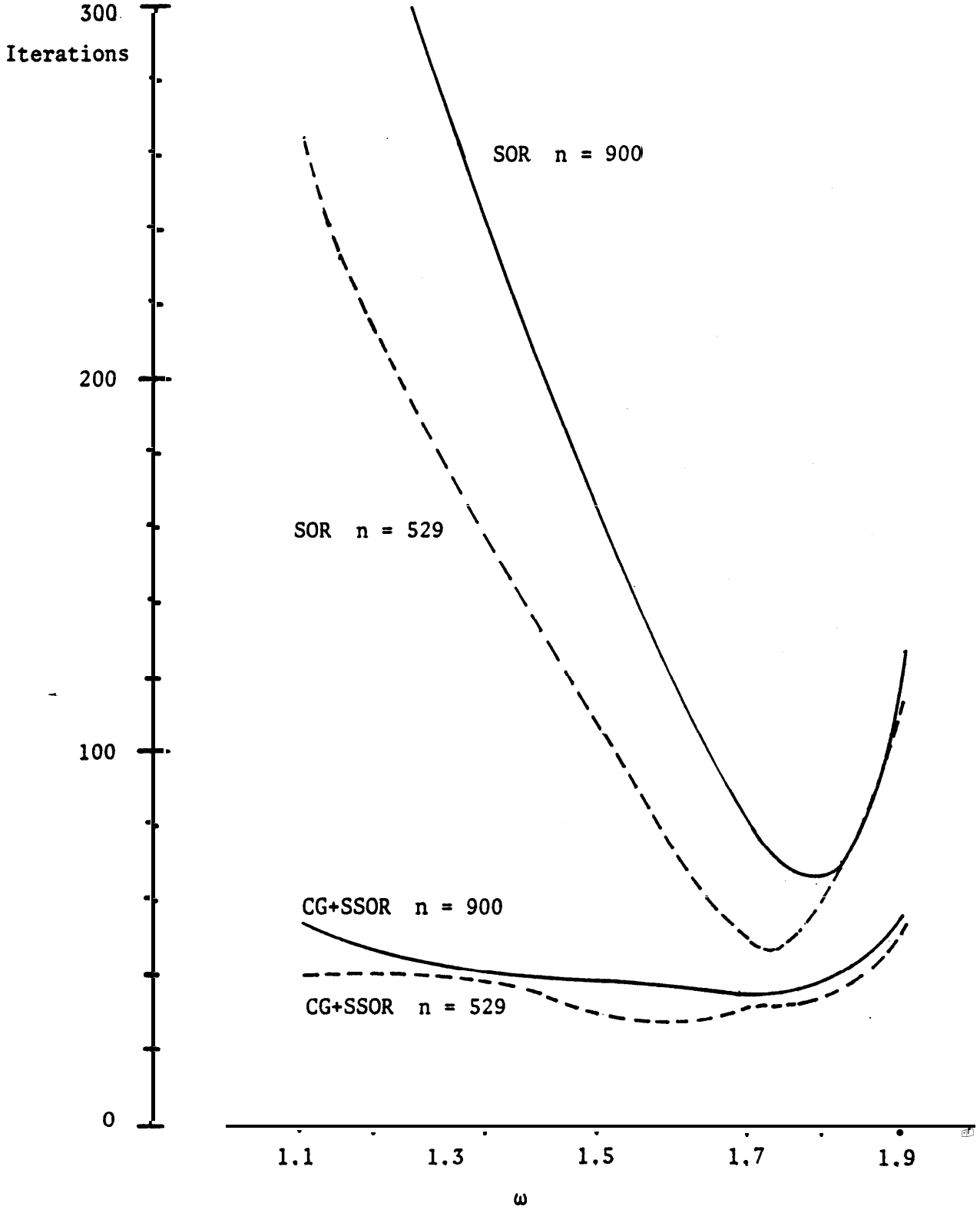


Figure 5. Algorithm Performance on Example 2 with Varying ω , C = 5.

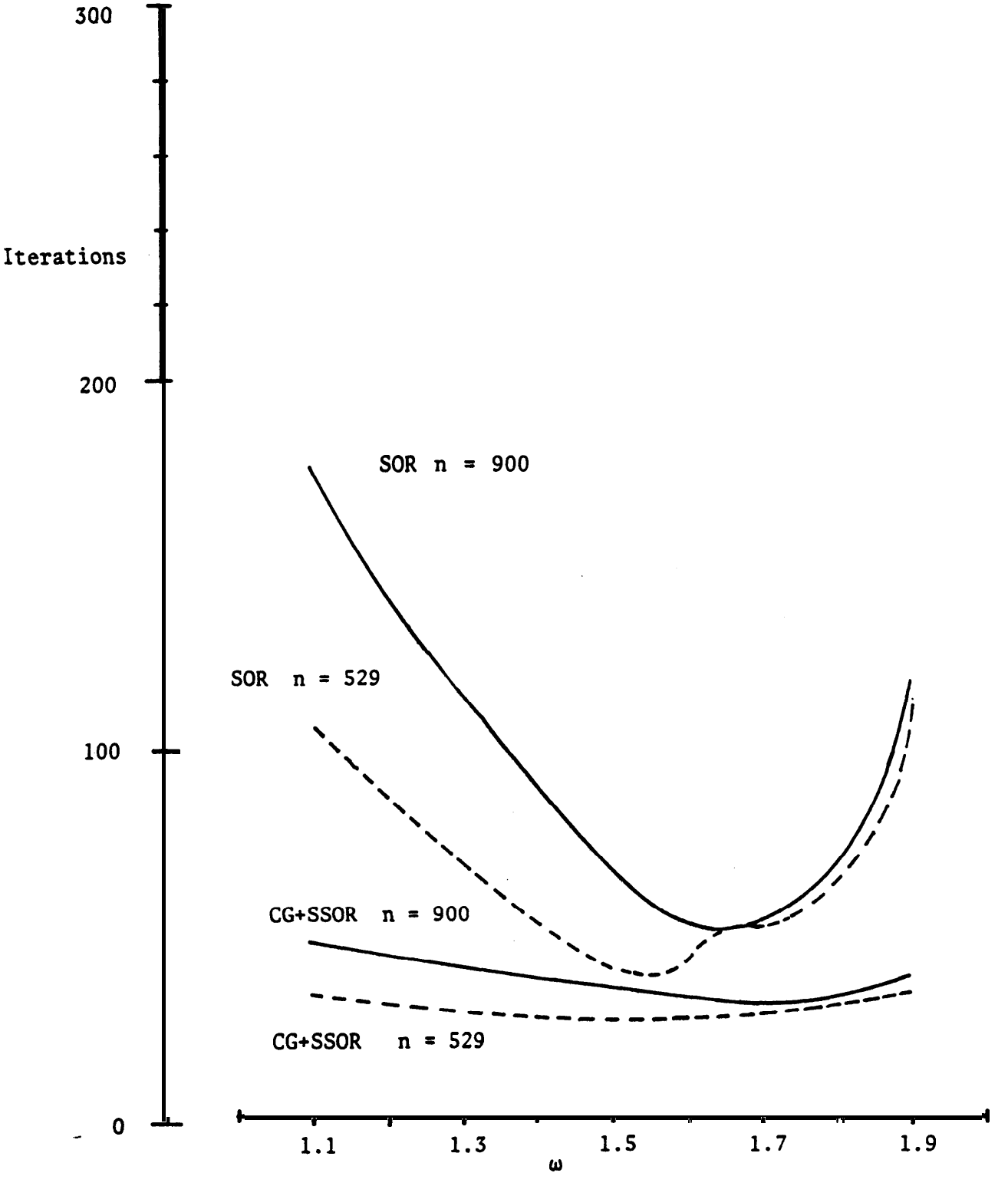


Figure 6. Algorithm Performance on Example 2 with Varying ω , C = 9.

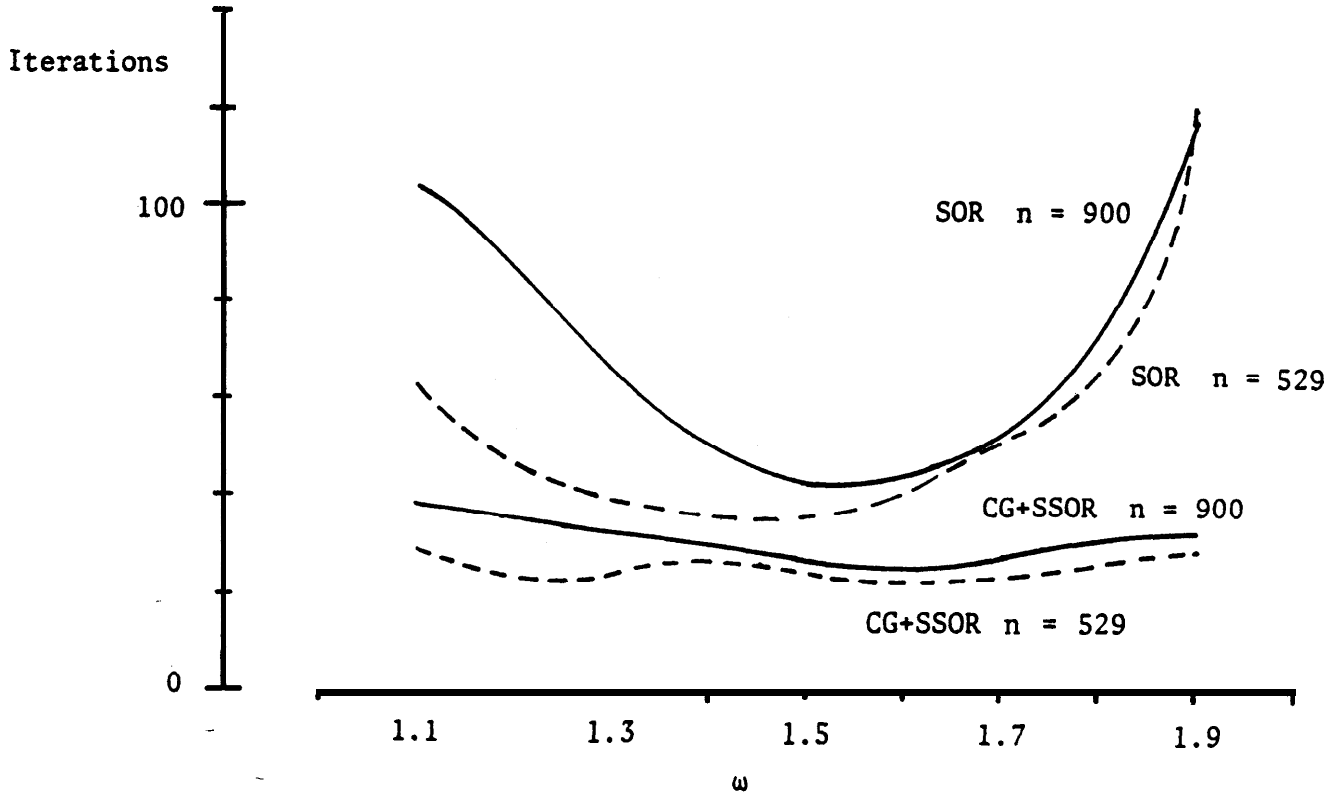


Figure 7. Algorithm Performance on Example 2 with Varying ω, C = 13.

The typical pattern for examples using conjugate gradients with SSOR scaling is that many restarts take place at the beginning until a reasonable set I is identified. Throughout this period then, the algorithm is equivalent to SSOR used alone with some variables kept fixed. Once I has stablized, few restarts occur, so the fast convergence of the conjugate gradient algorithm can be exploited with great effectiveness.. One of the advantages of this algorithm is that the transition from SSOR toconjugategradientswith SSOR scaling is is made autanatically.

Example 3

The matrix A of Examples 1 and 2 is a **2-cyclic** matrix, and theory tells us the optimal ω for the SOR iteration for a linear system. The matrix in this example is not **2-cyclic.** It is the discrete Laplacian g-point operator

$$A = \begin{pmatrix} T_1 & T_2 & & & \\ T_2 & T_1 & T_2 & & \\ & \ddots & \ddots & \ddots & \\ & & T_2 & T_1 & T_2 \\ & & & T_2 & T_1 \end{pmatrix}_{m^2 \times m^2} \qquad T_1 = \begin{pmatrix} 20 & -4 & & & \\ -4 & 20 & -4 & & \\ & \ddots & \ddots & \ddots & \\ & & -4 & 20 & -4 \\ & & & -4 & 20 \end{pmatrix}_{m \times m},$$

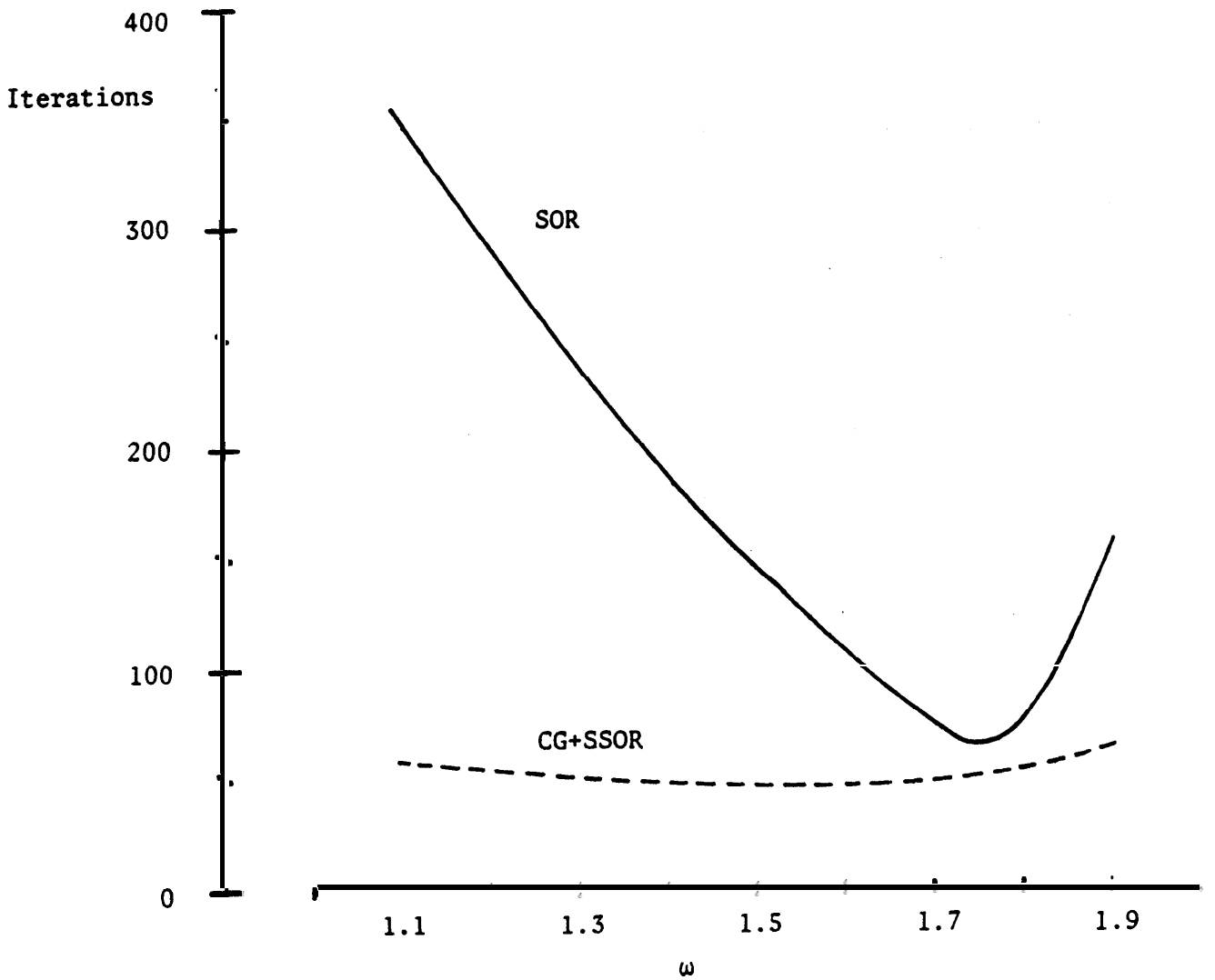


Figure 8. Algorithm Performance on Example 3 with Varying ω , n = 529.

$$T_2 = \begin{pmatrix} -4 & -1 & & & \\ -1 & -4 & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & -1 \\ & & & -1 & -4 \end{pmatrix}_{m \times m}$$

All other features of this example were the same as in Example 1. Figure 8 shows the results of applying the conjugate gradient algorithm with SSOR scaling and the SOR algorithm to a matrix of dimension n = 529 with five random vectors b . Results are similar to Example 1, with SOR showing sensitivity to ω while the number of iterations for conjugate gradients with SSOR scaling is relatively constant. The number of variables not at their bounds in the final solution varied from 513 to 463 for the five problems, and the number of outer iterations was 7 for all of the conjugate gradient runs.

4. Discussion and Conclusions

We have presented a conjugate gradient algorithm with matrix splittings which is suitable for certain quadratic programming problems. The performance of the method on special classes of problems might be enhanced by preprocessing or by modifications to the inner iteration.

For applications with A an M-matrix , the pre-processing scheme of Cottle and Goheen [5] could be used before beginning our algorithm in order to identify some of the variables which will be at their bounds in the optimal solution. These variables could then be held fixed throughout the conjugate gradient iteration.

Other algorithms could be substituted for the conjugate gradient iteration, as long as there is a descent function for the inner iteration which guarantees that no subproblem will repeat. The conjugate gradient algorithm is quite versatile, however, and has rapid convergence when used with a suitable scaling matrix. Such scalings may be chosen to be portions of the matrix A (for example, the diagonal or band part of the matrix) or an operator arising from application of an iterative method forsolving linear systems. Operators for related physical problems may also be used effectively. For example, a fast direct method for solving Laplace's equation over a regular region might be used as a scaling for a problem with a matrix corresponding to Laplace's equation over a region which does not permit separation of variables.

The conjugate gradient algorithm with matrix splittings has been demonstrated to have finite termination and to be effective for free boundary problems for elliptic partial differential equations. The method, however, requires only that the matrix A be positive definite and thus

has broader applications. Test results suggest that the algorithm is effective whether or not the constraints are tight.

Acknowledgements

Part of this work was completed while I was a doctoral student of Dr. Gene H. Golub at Stanford University. I am deeply grateful to him for his inspiration, guidance, and continual encouragement. This research was begun at his suggestion, and he has given valuable advice improving the work and its presentation. Special thanks go to Mr. Lee Zukowski who prepared the figures and to Mr. Franklin Luk for his careful reading of the manuscript.

REFERENCES

[1] O. Axelsson, "On preconditioning and convergence acceleration in sparse matrix problems", Report CERN 74-10, CERN European Organization for Nuclear Research (Geneva, 1974).

[2] C. Baiocchi, V. Comincioli, E. Magenes, and G.A. Pozzi, "Free boundary problems in the theory of fluid flow through porous media", Ann. Mat. Pura. Appl. 97 (1973) l-82.'

[3] J. Cea and R. Glowinski, "Sur des methodes d'optimisation par relaxation", R.A.I.R.O R-3 (1953) 5-32.

[4] Paul Concus, Gene H. Golub, and Dianne P. O'Leary, "A generalized conjugate gradient method for the numerical solution of elliptic partial differential equations", in: James R. Bunch and Donald J. Rose, ed., Sparse matrix computations (Academic Press, New York, 1976) pp. 309-332. `

[5] Richard W. Cottle and Mark S. Goheen, "A special class of iarge quadratic programs", Report SOL 76-7, Stanford University Systems Optimization Laboratory (Stanford, California, 1976).

[6] Richard W. Cottle, Gene H. Golub, and Richard Sacher, "On the solution of large, structured linear complementarity problems III", Report 74-7, Stanford University Operations Research Department (Stanford, California, 1974).

[7] C.W. Cryer, "The method of Christopherson for solving free boundary problems for infinite journal bearings by means of finite differences", Math. Comp. 25 (1971) 435-443.

[8] J.W. Daniel, "The conjugate gradient method for linear and nonlinear operator equations", SIAM J. Numer. Anal. 4 (1967) 10-26.

[9] G.B. Dantzig and R.W. Cottle, "Complementary pivot theory of mathematical programming", in: G.B. Dantzig and A.F. Veinott, Jr., ed., Mathematics of the decision sciences, part 1 (American Mathematical Society, Providence, R.I., 1968) pp. 115-136.

[10] Martin A. Diamond, "The solution of a quadratic programming problem using fast methods to solve systems of linear equations", Int. J. Systems Sci. 5 (1974) 131-136.

[11] R. Fletcher and M.P. Jackson, "Minimization of a quadratic function of many variables subject only to lower and upper bounds", J. Inst. Maths. Applics. 14(1974) 159-174.

[12] G.E. Forsythe and E.G. Straus, "On best conditioned matrices", Proc. Amer. Math. Soc. (1955) 340-345.

[13] G. Hadley, Nonlinear and dynamic programming (Addison-Wesley Publishing Co., Reading.Mass., 1964).

[14] Magnus R. Hestenes and Eduard Stiefel, "Methods of conjugate gradients for solving linear systems", J. Res. Nat. Bur. Standards 49(1952) 409-436.

[15] C.E. Lemke, "Bimatrix equilibrium points and mathematical programming", Management Sci. 11 (1965) 681-689.

[16] J.A. Meijerink and H.A. van der Vorst, "An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix", Math. Comp. 31 (1977) 148-162.

[17] W. Murray, "An algorithm for finding a local minimum of an indefinite quadratic program", Report NAC 1, National Physical Laboratory (Teddington, England, 1971).

[18] B.T. Polyak, "The conjugate gradient method in extremal problems", U.S.S.R. Computational Mathematics and Mathematical Physics 9 (1969) 94-112.

[19] Richard S. Sacher, 'On the solution of large, structured linear complementarity problems II", Report 73-5, Stanford University Operations Research Department (Stanford, California, 1974).

[20] J.H. Wilkinson, The algebraic eigenvalue problem (Clarendon Press), Oxford, 1965).