Computing 22, 59—77 (1979)

Computing

Conjugate Gradient Algorithms in the Solution of Optimization Problems for Nonlinear Elliptic Partial Differential Equations

Dianne P. O'Leary*, Ann Arbor, Michigan

Received July 12, 1978

Abstract — Zusammenfassung

Conjugate Gradient Algorithms in the Solution of Optimization Problems for Nonlinear Elliptic Partial Differential Equations. Several variants of the conjugate gradient algorithm are discussed with emphasis on determining the parameters without performing line searches and on using splitting techniques to accelerate convergence. The splittings used here are related to the nonlinear SSOR algorithm. The behavior of the methods is illustrated on a discretization of a nonlinear elliptic partial differential boundary value problem, the minimal surface equation. A conjugate gradient algorithm with splittings is also developed for constrained minimization with upper and lower bounds on the variables, and the method is applied to the obstacle problem for the minimal surface equation.

Konjugierte Gradienten-Algorithmen in der Lösung von Optimierungsproblemen für nichtlineare elliptische partielle Randwertprobleme. Wir besprechen mehrere Varianten des konjugierten Gradienten-Algorithmus unter Hervorhebung der Parameterbestimmung ohne Minimierung entlang von Linien und der Konvergenzbeschleunigung durch Zerlegung. Die hier verwendeten Zerlegungen sind dem nichtlinearen SSOR-Algorithmus verwandt. Das Verhalten der Methoden wird illustriert an der Diskretisierung eines nichtlinearen elliptischen partiellen Randwertproblems, nämlich der Minimalflächen-Gleichung. Wir entwickeln auch einen konjugierten Gradienten-Algorithmus mit Zerlegungen für Minimierung mit von oben und unten beschränkten Variabeln; ferner zeigen wir eine Anwendung der Methode auf das Hindernisproblem bei der Minimalflächen-Gleichung.

1. Introduction

We study in this paper the application of several variants of the conjugate gradient algorithm to the solution of large systems of nonlinear equations and certain optimization problems arising from discretization of nonlinear elliptic partial differential equations.

The conjugate gradient algorithm is an iterative method for solving certain systems of linear or nonlinear equations

$$g(u)=0$$

* This work was supported by the National Science Foundation under Grant MCS 76-06595 at the University of Michigan.

0010-485X/79/0022/0059/$ 03.80

or, alternatively, for finding a stationary point of a function $f(u)$ with gradient $g(u)$. Here u and g are n-vectors. The method was originally discussed [21] for convex quadratic functions:

$$f(u)=1/2\, u^T A u - u^T b$$
$$g(u)=Au-b,$$

where A is an $n\times n$ symmetric positive definite matrix. The strategy is, given an initial approximation $u^{(1)}$, to take steps that produce a sequence of iterates $\{u^{(k)}\}$ for which $\{f(u^{(k)})\}$ is a monotonically decreasing sequence. Each step direction is the negative of the component of the current gradient that is A-conjugate to all previous directions, and the function is minimized in each of these directions in turn. This method has several desirable properties: it terminates (under exact arithmetic) in at most n steps with u^*, the global minimizer of f; there are no arbitrary parameters to choose or extra information to provide; it requires only a few vectors of storage plus a means of forming the product of A with an arbitrary vector; and the convergence [10] is bounded as

$$E(u^{(k)})\leq 4\left(\frac{1-\sqrt{\kappa^{-1}}}{1+\sqrt{\kappa^{-1}}}\right)^{2k-2} E(u^{(1)}),$$

where κ is the condition number of A, defined to be its largest eigenvalue divided by its smallest eigenvalue, and $E(u)=f(u)+1/2\,(u^*, b)$. Because of the first property, we could consider the conjugate gradient algorithm to be a direct method for solving linear systems, but its operation count is too high for it to be generally useful in this mode. The last three properties are the reasons it has become a popular iterative method for solving large sparse systems of linear equations.

The algorithm has been generalized in many ways to solve nonlinear systems of equations. See, for example, [2, 10, 11, 13, 16, 17, 23, 26, 29, 30, 31]. These algorithms differ in their choice of step directions and the accuracy of the minimization required at each step. Some require information such as function evaluation and Jacobian matrices while others do not. They are alternatives to Newton's method when it is impractical to solve linear systems involving the Jacobian matrix of g. They are used instead of variable metric (quasi-Newton) methods [4] when the number of unknowns precludes storage and updating of an approximation to the Jacobian or its inverse. Conjugate gradient algorithms share many desirable properties with variable metric methods; in particular, under certain conditions on f and the choice of parameters [3, 5, 22, 24], these algorithms can be shown to have an n step superlinear or n step quadratic convergence rate: i.e.,

$$\limsup_{k\to\infty} \frac{\| u^{(k+n)}-u^*\|}{\| u^{(k)}-u^*\|^{\theta}}\leq c<\infty$$

where $\theta>1$ or $\theta=2$ respectively and c is a constant. Unfortunately, these convergence rate estimates are useless for our problems. The number of unknowns is so large that an algorithm must converge in much fewer than n iterations if it

is to be a feasible procedure for problems arising from discretization of partial differential equations.

Our study of the conjugate gradient algorithm is a continuation of work reported by Concus, Golub, and O'Leary [9]. In that paper, several versions of the conjugate gradient algorithm for convex functions were considered and practical criteria were developed to ensure convergence of the algorithm without requiring accurate minimization along the step directions. Applications to discretization of nonlinear elliptic differential equations were considered, and the nonlinear operators were split in order to improve convergence. These splittings were based on related elliptic operators, partial factorizations, or iterative methods such as symmetric successive overrelaxation (SSOR). Test results for the minimal surface equation and for a mildly nonlinear equation arising in the theory of semiconductor devices showed several conjugate gradient algorithms to be competitive with other algorithms on such problems.

In this paper we extend this work in several ways. In Section 2 we present the algorithms described in [9], adding a modification that reduces the necessity to restart those conjugate gradient algorithms in order to guarantee convergence. Numerical experiments applying this family of algorithms to the minimal surface equation are presented in Section 3. In Section 4 we discuss an algorithm applicable to minimization of a nonlinear function subject to upper and lower bounds on the variables, and in Section 5 present the results of numerical experiments on minimal surfaces with obstacles.

We will use the notation

$$\bar{\alpha} = \arg\min_{\alpha > 0} h(\alpha)$$

when $\bar{\alpha} > 0$ and $h(\bar{\alpha}) = \min_{\alpha > 0} h(\alpha)$.

2. Various Forms of the Conjugate Gradient Algorithm

The basic conjugate gradient algorithm for minimizing the convex function $f(u)$ with gradient $g(u)$ is as follows: Given an initial iterate $u^{(1)}$, set $r^{(1)} = -g(u^{(1)})$ and the initial direction $p^{(1)} = r^{(1)}$, and for $k = 1, 2, \ldots$, form

$$u^{(k+1)} = u^{(k)} + \alpha_k p^{(k)}$$

$$r^{(k+1)} = -g(u^{(k+1)})$$

$$p^{(k+1)} = r^{(k+1)} + \beta_k p^{(k)}.$$

The parameters α_k and β_k are chosen, in the convex quadratic case, to minimize f along the direction $p^{(k)}$ and to make $(p^{(k+1)}, A p^{(j)}) = 0$ for $j \leq k$:

$$\alpha_k = \arg\min_{\alpha > 0} f(u^{(k)} + \alpha p^{(k)}) = \frac{(r^{(k)}, r^{(k)})}{(p^{(k)}, A p^{(k)})} = \frac{(r^{(k)}, p^{(k)})}{(p^{(k)}, A p^{(k)})},$$

$$\beta_k = \frac{(r^{(k+1)}, r^{(k+1)})}{(r^{(k)}, r^{(k)})} = -\frac{(r^{(k+1)}, A p^{(k)})}{(p^{(k)}, A p^{(k)})} = \frac{(r^{(k+1)}, r^{(k+1)} - r^{(k)})}{(r^{(k)}, r^{(k)})}.$$

See [32] for derivations showing the equivalence of the various forms for the parameters. For convex nonquadratic problems, β_k is generally chosen by one of the three formulas above, where A is taken to be the Jacobian matrix $J(u^{(k)})$ or $J(u^{(k+1)})$. The parameter α_k is usually obtained by a line search procedure to be a good approximation to

$$\alpha_k^{opt} = \arg\min_{\alpha>0} f(u^{(k)} + \alpha\, p^{(k)}).$$

A scaled version of the conjugate gradient algorithm [1, 8, 10, 15, 19, 20, 25, 34] can be obtained in the quadratic case by applying the iteration formulas to the equivalent problem

$$\min_w \tilde{f}(w), \text{ where } \tilde{f}(w) = 1/2\, w^T M^{-1/2} A M^{-1/2} w - w^T M^{-1/2} b$$

and $M^{-1/2}$ is a symmetric positive definite matrix. If we then rewrite the formulas obtained in terms of the original variables $u = M^{-1/2} w$, we obtain the algorithm:

Given $u^{(1)}$, form $r^{(1)} = -g(u^{(1)})$, $z^{(1)} = M^{-1} r^{(1)}$, and $p^{(1)} = z^{(1)}$, and for $k = 1, 2, \ldots$ compute

$$u^{(k+1)} = u^{(k)} + \alpha_k\, p^{(k)}$$
$$r^{(k+1)} = -g(u^{(k+1)}),\ z^{(k+1)} = M^{-1} r^{(k+1)}$$
$$p^{(k+1)} = z^{(k+1)} + \beta_k\, p^{(k)}.$$

M is a positive definite matrix chosen to accelerate convergence. The parameters α_k and β_k are given by formulas $\alpha_k = \alpha_k^1 = \alpha_k^2 = \alpha_k^3$ and $\beta_k = \beta_k^1 = \beta_k^2 = \beta_k^3$ where

$$\alpha_k^1 = \frac{(r^{(k)}, z^{(k)})}{(p^{(k)}, A\, p^{(k)})}, \quad \beta_k^1 = \frac{(r^{(k+1)}, z^{(k+1)})}{(r^{(k)}, z^{(k)})},$$
$$\alpha_k^2 = \frac{(r^{(k)}, p^{(k)})}{(p^{(k)}, A\, p^{(k)})}, \quad \beta_k^2 = -\frac{(z^{(k+1)}, A\, p^{(k)})}{(p^{(k)}, A\, p^{(k)})}, \tag{2.1}$$
$$\alpha_k^3 = \arg\min_{\alpha>0} f(u^{(k)} + \alpha\, p^{(k)}), \quad \beta_k^3 = \frac{(r^{(k+1)}, z^{(k+1)} - z^{(k)})}{(r^{(k)}, z^{(k)})}.$$

In the nonquadratic case, these formulas are no longer equivalent but yield distinct conjugate gradient algorithms. Instead of the matrix A we use the Jacobian matrix $J_k = J(u^{(k)})$ in all parameters with subscript k. Fletcher and Reeves [16] proposed the parameter β_k^1, Daniel [10] used β_k^2 with $J(u^{(k+1)})$ rather than $J(u^{(k)})$, and β_k^3 was given by Polak and Ribiere [29].

We are interested in convex nonquadratic problems for which line searches are expensive but evaluation of the Jacobian matrix is feasible. In fact, for the discretization of nonlinear elliptic differential equations, it is desirable to avoid the need to calculate the function value $f(u^{(k)})$. If f is convex then $(p^{(k)}, g(u^{(k)} + \alpha\, p^{(k)}))$ is a monotone increasing function of α that is negative at $\alpha = 0$ and is zero at $\alpha = \alpha_k^3$, the point at which f attains its minimum on the line $u^{(k)} + \alpha\, p^{(k)}$. As long as our step underestimates α_k^3, we have guaranteed that $f(u^{(k+1)}) < f(u^{(k)})$ and

this is verified by checking that

$$(p^{(k)}, g^{(k+1)}) \leq 0. \tag{2.2}$$

We can define a conjugate gradient algorithm implementing this idea as follows:

Given $u^{(1)}$, the scaling operator M, a termination criterion ε, the index q of the β choice ($q=1, 2$, or 3), and K, the maximum number of iterations, let $r^{(1)} = -g(u^{(1)})$, $z^{(1)} = M^{-1} r^{(1)}$, and $p^{(1)} = z^{(1)}$.

Then for $k=1, 2, \ldots, K$:

(1) Choose α_k and calculate the new iterate $u^{(k+1)}$, the new residual $r^{(k+1)} = -g(u^{(k+1)})$, and the scaled residual $z^{(k+1)} = M^{-1} r^{(k+1)}$.

(2) Test for satisfaction of the termination criterion. This is usually a test that $\| r^{(k+1)} \| < \varepsilon$ for an appropriate norm.

(3) Update the direction to prepare for the next iteration:

$$p^{(k+1)} = z^{(k+1)} + \beta_k^q p^{(k)}$$

where β_k^q is defined by equation (2.1).

The algorithm is well defined except for Step (1). Here we modify the algorithm of [9]. The choice of α_k can be made as follows: We test candidates for α_k in turn until the convergence test (equation (2.2)) is satisfied. We test in the order α_k^1 (or α_k^2), then α_k^2 (or α_k^1). If both of these fail, a line search is performed until the test is satisfied. In our experiments, bisection was used for the line search, and the algorithm was restarted using the current vector $u^{(k)}$ as $u^{(1)}$ if two steps of bisection failed.

Each tentative choice of α_k involves computing

$$u_T^{(k+1)} = u^{(k)} + \alpha_k p^{(k)},$$

$$r_T^{(k+1)} = -g(u_T^{(k+1)})$$

and calculating $(r_T^{(k+1)}, p^{(k)})$. For the successful step, however, this extra inner product is essentially free, since $(r^{(k+1)}, z^{(k+1)})$ and $(r^{(k+1)}, p^{(k+1)})$ will be needed to calculate α_{k+1}^1 and α_{k+1}^2, and from the definition of $p^{(k+1)}$,

$$(r^{(k+1)}, p^{(k+1)}) = (r^{(k+1)}, z^{(k+1)}) + \beta_k^q (r^{(k+1)}, p^{(k)}). \tag{2.3}$$

For our problems, K is generally quite small ($K \approx 10$ when $n \approx 400$) and the conjugate gradient cycle is restarted from the current u whenever bisection fails or when more than K iterations have been performed in the current cycle. The addition of an accurate line search when $k=1$ would guarantee convergence for certain classes of functions f. The choice of the scaling operator M will be discussed in Section 3.

The elimination of accurate line searches in most iterations can produce a large savings in gradient calculations per iteration, but must be implemented carefully in order to avoid greatly increasing the number of iterations and thus the number of Jacobian evaluations.

As noted above, in the quadratic case, all choices of the parameters α_k and β_k are equivalent. The question arises, if in the course of the computation on a non-quadratic problem without line searches our calculated parameters agree (i.e., $\alpha_k^1 = \alpha_k^2$ and $\beta_k^1 = \beta_k^2 = \beta_k^3$), what can we conclude about the function?

Theorem:

(*i*) *The parameters* α_k^1 *and* α_k^2 *satisfy* $\alpha_k^1 = \alpha_k^2$ *if and only if* $(p^{(k-1)}, r^{(k)}) = 0$.

(*ii*) *If* α_k^1 *is used,* $\beta_k^1 = \beta_k^2$ *holds if and only if* $(z^{(k+1)}, r^{(k+1)} + \alpha_k^1 J_k p^{(k)}) = 0$.

(*iii*) *There holds* $\alpha_k^1 = \alpha_k^2$ *and* $\beta_k^1 = \beta_k^2 = \beta_k^3$ *if and only if* $(p^{(k-1)}, r^{(k)}) = 0$, $(r^{(k+1)}, z^{(k)}) = 0$, *and* $(z^{(k+1)}, r^{(k+1)} + \alpha_k^1 J_k p^{(k)}) = 0$.

Proof:

(i) This result follows from the definitions of α_k^1 and α_k^2 and equation (2.3).

(ii) This follows from the definition of α_k^1 and the observation that

$$\frac{-(z^{(k+1)}, J_k p^{(k)})}{(p^{(k)}, J_k p^{(k)})} = \frac{-(z^{(k+1)}, r^{(k+1)}/\alpha_k^1 + J_k p^{(k)}) + (r^{(k+1)}, z^{(k+1)})/\alpha_k^1}{(p^{(k)}, J_k p^{(k)})}.$$

(iii) This follows from parts (1) and (2).

Notice that $r^{(k+1)} + \alpha_k J_k p^{(k)}$ would be equal to $r^{(k)}$ if the function were quadratic.

These algebraic properties give the answer to our question: the parameters agree if and only if the line search at the previous step was exact and the new residual is M^{-1}-conjugate to the previous residual and to the approximation to it given by the quadratic theory. If this occurs for a full cycle of conjugate gradient steps and the size of the residual indicates that we are in the neighborhood of the solution, it would be desirable to delay the restart and permit the length of a cycle to be a full n or $n+1$ steps. If n is large, we would expect convergence to occur long before the cycle is complete.

3. Numerical Results

The various choices of parameters for the conjugate gradient algorithm were tested on a minimal surface problem. Other solution techniques for this problem can be found, for example, in [6, 7, 12, 18, 33]. The specific example used here is found in [6] and [9]. This is the minimal surface equation over a rectangular region:

$$\operatorname{div}(\gamma \nabla v) = 0 \quad \text{on} \quad R$$

$$R = \{(x, y) : 0 < x < 2,\ 0 < y < 1\}$$

where $\gamma = (1 + |\nabla v|^2)^{-1/2}$ and the boundary conditions are

$$v(0, y) = v(2, y) = v(x, 1) = 0$$

$$v(x, 0) = \sin \pi x/2.$$

The solution is symmetric about the line $x=1$, so the problem was solved over the unit square. A uniform mesh of size h ($h=1/s$) is imposed on the domain. The approximation to $v(mh, ih)$ is denoted by $u_{m,i}$. The finite difference equations are

$$g_{m,i}=\gamma_{\overline{m},\overline{i}}(2u_{m,i}-u_{m-1,i}-u_{m,i-1})+$$
$$+\gamma_{\overline{m+1},\overline{i}}(2u_{m,i}-u_{m+1,i}-u_{m,i-1})$$
$$+\gamma_{\overline{m},\overline{i+1}}(2u_{m,i}-u_{m-1,i}-u_{m,i+1})$$
$$+\gamma_{\overline{m+1},\overline{i+1}}(2u_{m,i}-u_{m+1,i}-u_{m,i+1})=0 \qquad m=1,2,\ldots,s-1,\ i=1,2,\ldots,s-1.$$

Here $\gamma_{\overline{m},\overline{i}}$ is γ at the point $((m-1/2)h, (i-1/2)h)$ using the approximation

$$|\nabla u|_{\overline{m},\overline{i}}^2=\frac{1}{2h^2}\left((u_{m,i}-u_{m-1,i})^2+(u_{m,i}-u_{m,i-1})^2\right.$$
$$\left.+(u_{m,i-1}-u_{m-1,i-1})^2+(u_{m-1,i}-u_{m-1,i-1})^2\right).$$

Appropriate modifications are made to the finite difference equation near the Neumann boundary $x=1$ ($m=s$). See [9] for details. The equations are second order accurate.

The finite difference equations have an alternate interpretation as the gradient of

$$f(u)=h^2\sum_{i=1}^{s}\sum_{m=1}^{s}\sqrt{1+|\nabla u|_{\overline{m},\overline{i}}^2}$$

which is an approximation to the surface area

$$S(v)=\int_0^1\int_0^1(1+v_x^2+v_y^2)^{1/2}\,dx\,dy.$$

The Jacobian matrix of g is sparse with at most 9 nonzero elements per row, so multiplication of an arbitrary vector with it is rather inexpensive though solving a linear system involving it would be costly.

Two scalings were used for the conjugate gradient algorithm; both are related to relaxation methods discussed, for example, in [28]. Consider first the one-step block successive overrelaxation-Newton (BSOR-Newton) algorithm. We partition u, r, and J into blocks corresponding to columns of mesh points. Then the new approximation to the i-th block of u, $u_i=(u_{1,i}, u_{2,i}, \ldots, u_{s,i})$, is obtained by the formula

$$u_i^{new}=u_i^{old}+\omega J_{i,i}^{-1}r_i \qquad i=1,2,\ldots,s-1.$$

Here r and J are evaluated at the most recent u values and ω is a scalar parameter. A symmetrized version of this algorithm, which obtains the next u vector by two steps of the algorithm, sweeping through the formulas from 1 to $s-1$ and then from $s-1$ to 1, will be called block symmetric successive over-relaxation-Newton (BSSOR-Newton) algorithm. This algorithm requires two Jacobian and gradient evaluations per iteration. When this method is used as a scaling, we define the vector $z^{(k)}$ to be the change in u over one double sweep of the algorithm starting from the guess $u^{(k)}$.

An alternate algorithm, the symmetric one-step Newton-BSSOR algorithm, requires only one gradient and Jacobian evaluation per iteration. Used as a scaling, this algorithm is defined by a forward and backward sweep as follows:

$$\bar{z}_i^{(k)} = \omega J_{i,i}^{-1} \left(r_i^{(k)} - (L\bar{z}^{(k)})_i\right), \quad i = 1, 2, \ldots, s-1$$

$$z_i^{(k)} = \bar{z}_i^{(k)} + \omega J_{i,i}^{-1} \left(r_i^{(k)} - (L\bar{z}^{(k)} + D\bar{z}^{(k)} + Uz^{(k)})_i\right), \quad i = s-1, s-2, \ldots, 2, 1,$$

where $J(u^{(k)})$ is partitioned as $L+D+U$, with D the block diagonal, L the block strictly lower triangular part, and U the block strictly upper triangular part. J and r in the formulas are evaluated at $u^{(k)}$.

See [9] for further description of the use of these algorithms as scalings.

In the tables we list the number of gradient and Jacobian evaluations necessary to obtain a residual with infinity norm less than ε for the various conjugate gradient algorithms and for the BSOR-Newton method. The mesh size used in the experiments was $h = 1/20$ (380 unknowns in the half domain) or $h = 1/40$ (1560 unknowns). The initial guess was $u^{(1)} = 0$, giving $\| r^{(1)} \|_\infty = .31$. The BSOR-Newton algorithm requires one gradient and one Jacobian evaluation per iteration. The conjugate gradient (CG) algorithm with Newton-BSSOR scaling takes one gradient and one Jacobian evaluation per iteration, plus an additional gradient evaluation for each extra α. The CG algorithm with BSSOR-Newton scaling requires three gradient and three Jacobian evaluations per iteration plus an additional gradient for each extra α. The test for an acceptable α, equation (2.2), was weakened slightly to require only that $(p^{(k)}, g^{(k+1)}) \leq \varepsilon \| g^{(k+1)} \|_\infty^2$.

Table 1 presents the results for the three algorithms using various values of ω.

For the CG algorithms, the parameters $\beta^{(2)}$, $K = 10$, and a first choice of $\alpha^{(2)}$ were used. The ranges and average amounts of work are summarized in Table 2. We note that BSOR-Newton had the greatest variation in performance with ω, while the CG algorithms were much less sensitive. For low accuracy, $\varepsilon \geq 10^{-2}$, the BSOR-Newton algorithm was usually less expensive than the CG algorithms, but for $\varepsilon \leq 10^{-3}$ the CG algorithms showed advantages. The CG algorithm with Newton-BSSOR scaling required far fewer Jacobian evaluations and a comparable or fewer number of gradient evaluations than BSOR-Newton. The BSSOR-Newton scaled CG algorithm was also often less expensive than BSOR-Newton, although the Newton-BSSOR scaling was clearly more effective. The various α and β parameters quite often agreed for the BSSOR-Newton scaling, but less often with the Newton-BSSOR scaling. Results presented in Table 3 indicate that $\alpha^{(2)}$ is a poor choice of parameter, so we expect that the CG algorithm would be even more effective using $\alpha^{(1)}$.

Table 3 presents the results of using the CG algorithm with Newton-BSSOR scaling and $\omega = 1.6$ for various choices of the α, β, and restart parameters. The algorithm was relatively insensitive to restart values K between 5 and 15. Experiments indicated that 9 was optimal, and it was better to underestimate rather than overestimate this parameter. When using $\alpha^{(1)}$, the number of gradient and Jacobian evaluations was less than, or at worst 2 more than, the number when using $\alpha^{(2)}$. The choice of α had little effect on the number of Jacobian evaluations

Table 1. *Number of gradient and Jacobian evaluations needed to attain a residual* $\| r^{(k)} \|_\infty < \varepsilon$ *on test problem with* $h = 1/20$

Algorithm	ω	$\varepsilon = 10^{-6}$	$\varepsilon = 10^{-4}$	$\varepsilon = 10^{-2}$
BSOR-Newton	1.1	>200, >200	139, 139	14, 14
	1.2	>200, >200	113, 113	12, 12
	1.3	178, 178	91, 91	11, 11
	1.4	141, 141	72, 72	10, 10
	1.5	108, 108	56, 56	9, 9
	1.6	79, 79	43, 43	9, 9
	1.7	51, 51	31, 31	9, 9
	1.8	52, 52	31, 31	12, 12
	1.9	112, 112	68, 68	21, 21
CG+Newton-BSSOR $\alpha^{(2)}, \beta^{(2)}, K=10$	1.1	74, 34	57, 26	31, 14
	1.2	59, 30	38, 18	29, 13
	1.3	60, 26	49, 20	32, 13
	1.4	81, 31	59, 23	29, 11
	1.5	64, 26	42, 18	18, 8
	1.6	54, 23	40, 17	27, 11
	1.7	75, 34	54, 25	26, 11
	1.8	69, 33	42, 20	33, 14
	1.9	113, 44	67, 27	33, 14
CG+BSSOR-Newton $\alpha^{(2)}, \beta^{(2)}, K=10$	1.1	89, 75	70, 57	30, 24
	1.2	121, 90	72, 54	31, 24
	1.3	90, 72	66, 51	31, 24
	1.4	78, 63	58, 48	31, 24
	1.5	78, 60	66, 48	24, 18
	1.6	101, 69	66, 45	38, 27
	1.7	109, 78	77, 57	28, 21
	1.8	85, 66	52, 42	31, 24
	1.9	154, 117	116, 87	93, 69

Table 2. *Summary of number of gradient and Jacobian evaluations needed for the algorithms in Table* 1

	$\varepsilon = 10^{-6}$	$\varepsilon = 10^{-4}$	$\varepsilon = 10^{-2}$
BSOR-Newton			
minimum	51, 51	31, 31	9, 9
maximum	>200, >200	139, 139	21, 21
average	>125, >125	72, 72	12, 12
CG+Newton-BSSOR			
minimum	54, 23	38, 17	18, 8
maximum	113, 44	67, 27	33, 14
average	72, 31	50, 22	29, 12
CG+BSSOR-Newton			
minimum	78, 60	52, 42	24, 18
maximum	154, 117	116, 87	93, 69
average	101, 77	71, 54	37, 28

(i.e., the number of iterations), but the number of gradient evaluations was often significantly lower with $\alpha^{(1)}$. Among the β parameters, $\beta^{(3)}$ seemed better than $\beta^{(1)}$ and $\beta^{(2)}$, but when K was small there was not much difference in performance. On this problem, the combination $\alpha^{(1)}$ and $\beta^{(3)}$ gave a number of Jacobian evaluations within 1 of the minimum for each set of experiments. The combination $\alpha^{(1)}$, $\beta^{(1)}$ was always within 2 of the minimum for gradient evaluations, while $\alpha^{(1)}$, $\beta^{(2)}$ was competitive but usually not as effective as using $\beta^{(1)}$ or $\beta^{(3)}$.

When $\alpha^{(1)}$ was used, the line search was invoked very little, and then usually only at the first iterations. The first choice of $\alpha^{(2)}$ failed more often and when it did, $\alpha^{(1)}$ usually did, too. When this happened there were at least 3 trial α's, driving up the number of gradient evaluations.

Table 3 also presents data for $\omega = 1.2$, $K = 10$, and various choices of α and β. This shows that the conclusions regarding the best choices of α and β are valid for a range of ω values.

Further experiments indicated that the CG algorithm with Newton-BSSOR scaling was almost always more expensive than the BSSOR-Newton scaling in number of gradient and Jacobian evaluations. Both algorithms were insensitive to the magnitude of the boundary conditions.

Table 3. *Number of gradient and Jacobian evaluations needed for the CG algorithm with Newton-BSSOR scaling,* $h = 1/20$

		$\varepsilon = 10^{-6}$	$\varepsilon = 10^{-4}$	$\varepsilon = 10^{-2}$
$\omega = 1.6$, $K = 5$	$\alpha^{(2)}$, $\beta^{(1)}$	67, 32	45, 21	25, 12
	$\alpha^{(2)}$, $\beta^{(2)}$	57, 23	44, 18	24, 10
	$\alpha^{(2)}$, $\beta^{(3)}$	56, 24	42, 18	25, 11
	$\alpha^{(1)}$, $\beta^{(1)}$	27, 23	21, 17	14, 12
	$\alpha^{(1)}$, $\beta^{(2)}$	46, 24	40, 18	24, 10
	$\alpha^{(1)}$, $\beta^{(3)}$	45, 22	37, 18	25, 11
$\omega = 1.6$, $K = 10$	$\alpha^{(2)}$, $\beta^{(1)}$	47, 30	38, 24	27, 15
	$\alpha^{(2)}$, $\beta^{(2)}$	54, 23	40, 17	27, 11
	$\alpha^{(2)}$, $\beta^{(3)}$	62, 24	48, 19	27, 11
	$\alpha^{(1)}$, $\beta^{(1)}$	31, 27	27, 23	19, 15
	$\alpha^{(1)}$, $\beta^{(2)}$	44, 23	35, 17	27, 11
	$\alpha^{(1)}$, $\beta^{(3)}$	40, 23	34, 17	26, 11
$\omega = 1.6$, $K = 15$	$\alpha^{(2)}$, $\beta^{(1)}$	64, 36	42, 26	35, 20
	$\alpha^{(2)}$, $\beta^{(2)}$	57, 24	47, 19	25, 11
	$\alpha^{(2)}$, $\beta^{(3)}$	77, 30	56, 22	27, 11
	$\alpha^{(1)}$, $\beta^{(1)}$	39, 35	27, 25	20, 18
	$\alpha^{(1)}$, $\beta^{(2)}$	51, 24	46, 19	26, 11
	$\alpha^{(1)}$, $\beta^{(3)}$	38, 20	33, 15	26, 11
$\omega = 1.2$, $K = 10$	$\alpha^{(2)}$, $\beta^{(1)}$	63, 36	46, 26	25, 14
	$\alpha^{(2)}$, $\beta^{(2)}$	59, 30	38, 18	29, 13
	$\alpha^{(2)}$, $\beta^{(3)}$	42, 21	37, 18	24, 11
	$\alpha^{(1)}$, $\beta^{(1)}$	39, 35	29, 27	12, 12
	$\alpha^{(1)}$, $\beta^{(2)}$	48, 27	37, 20	30, 13
	$\alpha^{(1)}$, $\beta^{(3)}$	37, 21	31, 17	22, 11

Table 4. *Number of gradient and Jacobian evaluations on test problem with* $h = 1/40$

Algorithm	ω	$\varepsilon = 10^{-6}$	$\varepsilon = 10^{-4}$	$\varepsilon = 10^{-2}$
BSOR-Newton	1.2	>400, >400	321, 321	5, 5
	1.3	>400, >400	259, 259	4, 4
	1.4	>400, >400	206, 206	4, 4
	1.5	377, 377	160, 160	4, 4
	1.6	281, 281	120, 120	5, 5
	1.7	196, 196	86, 86	6, 6
	1.8	120, 120	58, 58	7, 7
	1.9	118, 118	74, 74	28, 28
CG+Newton-BSSOR $\alpha^{(2)}, \beta^{(2)}, K=10$	1.2	126, 58	90, 41	39, 18
	1.3	132, 55	100, 39	17, 6
	1.4	126, 54	69, 31	10, 6
	1.5	117, 53	83, 35	22, 10
	1.6	117, 49	87, 36	6, 4
	1.7	100, 43	75, 33	24, 11
	1.8	86, 38	72, 30	27, 12
	1.9	84, 38	67, 29	42, 18
CG+BSSOR-Newton $\alpha^{(2)}, \beta^{(2)}, K=10$	1.2	171, 135	120, 93	50, 39
	1.3	212, 153	124, 93	40, 30
	1.4	169, 129	122, 93	75, 57
	1.5	148, 117	103, 81	49, 42
	1.6	205. 150	134, 99	83, 66
	1.7	144, 102	110, 78	46, 33
	1.8	131, 87	107, 69	51, 33
	1.9	>400, >400	>400, >400	>400, >400

Table 5. *Summary of number of gradient and Jacobian evaluations needed for the algorithms in Table* 4

	$\varepsilon = 10^{-6}$	$\varepsilon = 10^{-4}$	$\varepsilon = 10^{-2}$
BSOR-Newton			
minimum	118, 118	58, 58	4, 4
maximum	>400, >400	321, 321	28, 28
average	>287, >287	161, 161	8, 8
CG+Newton-BSSOR			
minimum	84, 38	67, 29	6, 4
maximum	132, 58	100, 41	42, 18
average	111, 49	80, 34	23, 11
CG+BSSOR-Newton			
minimum	131, 87	103, 69	40, 30
maximum	>400, >400	>400, >400	>400, >400
average	>176, >141	>136, >112	>88, >78

Table 6. *Number of gradient and Jacobian evaluations needed for the CG algorithm with Newton-BSSOR scaling,* $h = 1/40$

		$\varepsilon = 10^{-6}$	$\varepsilon = 10^{-4}$	$\varepsilon = 10^{-2}$
$\omega = 1.6$, $K = 5$	$\alpha^{(2)}$, $\beta^{(1)}$	95, 46	61, 30	13, 7
	$\alpha^{(2)}$, $\beta^{(2)}$	99, 42	78, 33	6, 4
	$\alpha^{(2)}$, $\beta^{(3)}$	93, 42	72, 31	16, 8
	$\alpha^{(1)}$, $\beta^{(1)}$	51, 41	38, 30	24, 18
	$\alpha^{(1)}$, $\beta^{(2)}$	85, 44	72, 33	7, 4
	$\alpha^{(1)}$, $\beta^{(3)}$	81, 40	61, 26	26, 11
$\omega = 1.6$, $K = 15$	$\alpha^{(2)}$, $\beta^{(1)}$	89, 54	80, 46	35, 17
	$\alpha^{(2)}$, $\beta^{(2)}$	77, 38	67, 30	6, 4
	$\alpha^{(2)}$, $\beta^{(3)}$	79, 37	63, 29	44, 18
	$\alpha^{(1)}$, $\beta^{(1)}$	64, 54	57, 47	47, 37
	$\alpha^{(1)}$, $\beta^{(2)}$	85, 43	76, 34	7, 4
	$\alpha^{(1)}$, $\beta^{(3)}$	74, 39	60, 29	31, 13
$\omega = 1.4$, $K = 10$	$\alpha^{(2)}$, $\beta^{(1)}$	100, 54	75, 40	51, 27
	$\alpha^{(2)}$, $\beta^{(2)}$	134, 59	93, 41	10, 6
	$\alpha^{(2)}$, $\beta^{(3)}$	100, 49	63, 33	41, 19
	$\alpha^{(1)}$, $\beta^{(1)}$	60, 50	46, 38	11, 11
	$\alpha^{(1)}$, $\beta^{(2)}$	87, 48	68, 34	11, 6
	$\alpha^{(1)}$, $\beta^{(3)}$	71, 42	63, 34	19, 8
$\omega = 1.8$, $K = 10$	$\alpha^{(2)}$, $\beta^{(1)}$	93, 49	80, 42	64, 31
	$\alpha^{(2)}$, $\beta^{(2)}$	90, 40	75, 32	27, 12
	$\alpha^{(2)}$, $\beta^{(3)}$	93, 38	79, 30	28, 11
	$\alpha^{(1)}$, $\beta^{(1)}$	77, 59	65, 49	52, 38
	$\alpha^{(1)}$, $\beta^{(2)}$	78, 37	67, 30	29, 12
	$\alpha^{(1)}$, $\beta^{(3)}$	78, 36	71, 29	52, 21

An advantage of the CG algorithms over BSOR-Newton is that, because of their ability to incorporate adaptively a line search when necessary, they have better behavior when far away from the solution. This is not observed in the current implementation because of the relaxed condition for the downhill test, eqn. (2.2). With appropriate modification of this, however, the CG algorithms would exhibit a much larger practical radius of convergence than BSOR-Newton.

Tables 4—6 give results for a smaller mesh size, $h = 1/40$.

The trends in the first three tables are continued here. The CG algorithm with BSSOR-Newton scaling was superior to most BSOR-Newton runs for $\varepsilon \leq 10^{-3}$. Performance was not very sensitive to the choice of ω or K. For $\varepsilon \leq 10^{-4}$, $\alpha^{(1)}$, $\beta^{(3)}$ always gave results within 2 of the minimum for Jacobian evaluations, and $\alpha^{(1)}$, $\beta^{(1)}$ was within 2 of the minimum for gradient evaluations. The performance of $\alpha^{(1)}$, $\beta^{(2)}$ was variable.

Comparing Tables 2 and 5 for $\varepsilon \leq 10^{-4}$ we see that the number of BSOR-Newton gradient and Jacobian evaluations for the best ω doubled as h went from 1/20 to 1/40, and for CG with Newton-BSSOR scaling the growth factor was between 1.6 and 1.8.

From this limited data it would seem that $\alpha^{(1)}$, $\beta^{(3)}$ and $\alpha^{(1)}$, $\beta^{(1)}$ give the most effective CG algorithms, and that a choice between these would depend on the relative costs of gradient and Jacobian evaluations.

4. Algorithms for Constrained Problems

In this section we consider the solution of a convex minimization problem

$$\min f(u)$$

$$c \leq u \leq d.$$

We use the same notation as in Section 2, and modify the algorithms given there to solve this problem. The basic framework we use is due to Polyak [30].

The optimality conditions for the solution to the problem are

$$\text{If } u_i = c_i \text{ then } r_i \leq 0.$$

$$\text{If } u_i = d_i \text{ then } r_i \geq 0.$$

$$\text{If } c_i < u_i < d_i \text{ then } r_i = 0.$$

Our strategy will be to start with a $u^{(1)}$ such that $c \leq u^{(1)} \leq d$, and maintain feasibility, $c \leq u^{(k)} \leq d$, while iterating to satisfy the sign conditions on r. At each outer iteration we will choose a subset of the variables which are at their bounds and whose r components have the proper sign. We keep these u variables fixed while adjusting the others in an inner iteration.

In the algorithm, I is the index set for the fixed variables. For any vector w we denote by $\tilde{w}$ the vector obtained by setting the components of w corresponding to elements in the set I to zero:

$$\tilde{w}_i = \begin{cases} w_i & \text{if } i \notin I \\ 0 & \text{if } i \in I. \end{cases}$$

We present the algorithm and then make several comments on its relation to other algorithms and its convergence.

Given an initial feasible $u^{(1)}$, $c \leq u^{(1)} \leq d$, a scaling operator M, a termination criterion ε, the index q of the β choice, and K, the number of iterations in a cycle, set $r^{(1)} = -g(u^{(1)})$, and initialize $I = \{1, 2, \ldots, n\}$ so that the first termination test works.

(1) Let $I = \{i : u_i^{(1)} < c_i + \varepsilon \text{ and } r_i^{(1)} < 0\} \cup \{i : u_i^{(1)} > d_i - \varepsilon \text{ and } r_i^{(1)} > 0\}$. If I has not changed from the previous iteration and $\| \tilde{r}^{(1)} \| < \varepsilon$ then terminate with the solution. Otherwise, we will solve a subproblem keeping the elements of u corresponding to indices in I constant.

(2) Set $z^{(1)} = \tilde{r}^{(1)}$, $p^{(1)} = z^{(1)}$, $k1 = 1$, and begin Step 4. This initializes a steepest descent step followed by a scaled conjugate gradient cycle.

(3) Set $u^{(2)} = u^{(k)}$, $r^{(2)} = r^{(k)}$, $z^{(2)} = \widetilde{(M^{-1} r^{(2)})}$, $p^{(2)} = z^{(2)}$, and $k1 = 2$. This initializes a scaled conjugate gradient cycle.

(4) For $k = k1, \ldots, K$.

(a) If $\| \tilde{r}^{(k)} \| < \varepsilon$ then set $u^{(1)} = u^{(k)}$, $r^{(1)} = r^{(k)}$, and go to Step (1). We have successfully solved a subproblem.

(b) Calculate α_k^1 and α_k^2 defined in Section 2, and the maximum feasible step length in direction $p^{(k)}$:

$$\alpha_{\max} = \min \left(\min_{p_i^{(k)} > 0} \frac{d_i - u_i^{(k)}}{p_i^{(k)}}, \min_{p_i^{(k)} < 0} \frac{c_i - u_i^{(k)}}{p_i^{(k)}} \right)$$

Choose the step length α_k as in Section 2, but consider candidates only if they are less than $\alpha_{\max}$. If $k = 1$, continue the line search until success as defined by equation (2.2). If $k = 2$ and the search fails, set $u^{(1)} = u^{(2)}$, $r^{(1)} = r^{(2)}$, and go to Step (2). Otherwise, if the search fails, go to Step (3). If a successful α_k is found, set

$$u^{(k+1)} = u^{(k)} + \alpha_k p^{(k)}$$
$$r^{(k+1)} = -g(u^{(k+1)})$$

(c) Set $I = \{i : u_i^{(k+1)} < c_i + \varepsilon\} \cup \{i : u_i^{(k+1)} > d_i - \varepsilon\}$. If $I = \{1, 2, \ldots, n\}$ then go to Step (1) with $u^{(1)} = u^{(k+1)}$, $r^{(1)} = r^{(k+1)}$. If I has changed, then replace k by $k+1$ and go to Step (3) to restart the scaled algorithm.

(d) Calculate $z^{(k+1)} = (\widetilde{M^{-1} r^{(k+1)}})$. Update the direction to prepare for the next iteration:

$$p^{(k+1)} = z^{(k+1)} + \beta_k^q p^{(k)}$$

where β_k^q, $k \neq 1$, is defined in equation (2.1) and $\beta_1^q = 0$.

(5) Go to Step (2) to continue the solution process for the subproblem with $u^{(1)} = u^{(K+1)}$ and $r^{(1)} = r^{(K+1)}$.

Note first that if $c = -\infty$ and $d = \infty$, i.e., if the problem is unconstrained, then $I = \phi$ after Step (1) and the algorithm reduces to that of Section 2 modified to include occasional steepest descent steps.

Polyak considered the algorithm with M equal to the identity matrix and Step (2) omitted. If M is not the identity it is necessary to follow each definition of I in Step (1) by an unscaled steepest descent step in order to assure that the step direction is feasible. Although $r^{(1)}$ points away from the boundary, the scaled gradient $z^{(1)}$ may not. For suitable classes of functions f, if a more restrictive downhill test is used as in [22] or [24], the algorithm can be shown to converge. Each inner iteration (2)—(5) is just a restarted scaled conjugate gradient algorithm applied to a convex function with steepest descent steps interspersed. The algorithm proceeds in the normal way unless a constraint is encountered, in which case the number of variables is reduced and a new inner iteration is begun. The function f decreases at each iteration.

The use of this algorithm for quadratic objective functions is discussed in [27]. For constrained problems, the scaled vectors z are often truncated. For the Newton-BSSOR algorithm, for example, as $\bar{z}$ and z are computed, they are modified if necessary so that $u^{(k)} + \bar{z}^{(k)}$ and $u^{(k)} + z^{(k)}$ are feasible points.

In practice, the solution process is often more effective if a relatively large value of ε is used throughout the course of the iteration and then the algorithm is restarted with the desired smaller ε.

5. Numerical Results

The algorithms of Section 4 were applied to the minimal surface problem with obstacles:

$$\min S(v)$$

$$c(x, y) \leq v \leq d(x, y)$$

where S was defined in Section 3. Alternate approaches to this problem are given, for example, in [14]. The test problem was the same as in Section 3 with $d = \infty$ and various lower obstacles.

For the first set of tests, a lower obstacle was constructed that had a peak of C along the line $(\frac{1}{2}, \frac{1}{2})$ to $(1\frac{1}{2}, \frac{1}{2})$ and rectangular contours decreasing to zero at the boundary of the region R. The algebraic definition is

$$c(x, y) = 2C \min(x, 1/2 - |y - 1/2|)$$

for $0 \leq x \leq 1$ and $0 \leq y \leq 1$. The solution u for $C = 1$ is shown in Fig. 1. For clarity of display, the origin was taken in the right foreground and the region was reflected around the y axis.

The one step BSSOR-Newton algorithm was used as the scaling algorithm for this problem.

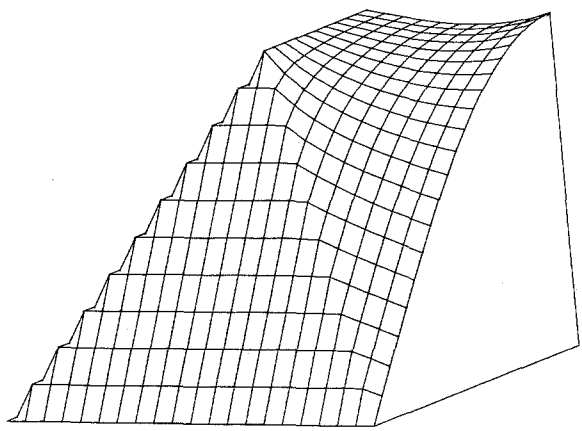

Fig. 1. Minimal surface for the first obstacle

Table 7. *Number of gradient and Jacobian evaluations needed for the constrained problem*

ω	$C=.3$	$C=1.0$
1.1	192, 126	245, 163
1.2	190, 126	232, 157
1.3	186, 124	227, 153
1.4	187, 125	242, 161
1.5	181, 121	236, 157
1.6	181, 119	237, 160
1.7	182, 128	223, 157
1.8	202, 136	241, 175
1.9	233, 161	294, 216

Table 7 shows the results of running the algorithm with various ω values using $u=c$ as the starting guess and a mesh size of $h=1/20$. The parameters $\alpha^{(1)}$ and $\beta^{(1)}$ were used and ε was taken to be 10^{-3} for initial convergence, with subsequent refinement to 10^{-6}. The performance was sensitive to the initial ε but not to the final one.

The counts of gradient and Jacobian evaluations shown in the table are a misleading estimate of the work, since in the course of the iteration, many variables were in the index set I and thus not all elements of u, g, and J need to be evaluated. For $C=.3$, for example, the number of variables in I stepped down monotonically from 238 to 11 over the course of 34—38 outer iterations. For $C=1$, the number of variables decreased monotonically from 167 to 29 over 35—38 iterations.

Other examples were designed to test the use of a starting guess more suitable than $u=c$. The problem was solved with $C=1$ and this solution was used as a starting guess for the problem with $C=.5$. Using $\omega=1.6$, 45 gradient and 39 Jacobian evaluations were needed for the second problem, rather than 183 and 131 respectively when starting from $u=c$. A series with C increasing from 0 to .3 in steps of .1 exhibited similar behavior. Thus significant saving can be expected if a parametric series of problems is to be solved.

Other experiments were performed with $h=1/20$ and point obstacles at the locations $(\frac{1}{2}, \frac{1}{2})$ and $(1\frac{1}{2}, \frac{1}{2})$. The parameters α^1, β^1, and ε were the same as in the previous run, and $\omega=1.6$. When the height of the obstacle was $C=.5$, the algorithm took 21 outer iterations with 190 gradient evaluations and 140 Jacobian evaluations from an initial guess of $u=0$. The height was then increased in steps of .5 up to 5, and the iteration was restarted from the previous solution with the value at $(\frac{1}{2}, \frac{1}{2})$ adjusted. In each case the new solution was found within 9—13 gradient and Jacobian evaluations. However, the surface did not change much in this example as the height of the obstacle was changed. The solution for $C=1$ is shown in Fig. 2.

If a solution on a fine mesh is required, it is far more efficient to obtain a solution on a coarse mesh and interpolate to obtain a starting guess, rather than solving

the fine mesh problem directly. In this way a good approximation to the correct set I is obtained and many restarts are avoided.

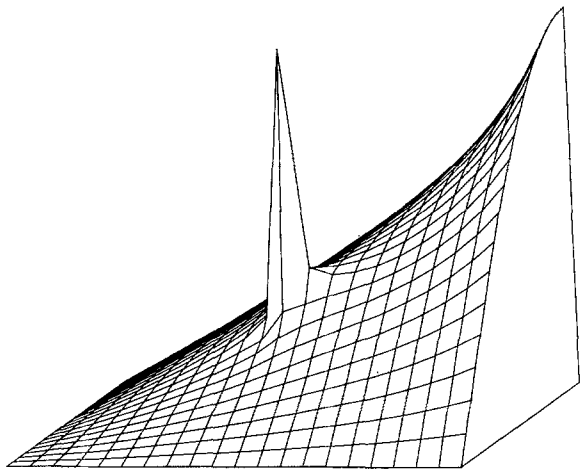

Fig. 2. Minimal surface for the second obstacle

6. Conclusions

The conjugate gradient algorithms described in this paper, scaled using relaxation methods, are practical and robust techniques for the solution of nonlinear equations and constrained minimization problems arising from discretization of nonlinear elliptic partial differential equations. In using these algorithms, the minimized function f never needs to be evaluated. The algorithms are simple to program, requiring less than 250 lines of FORTRAN code to implement the constrained algorithm and the scaling, plus code to evaluate the gradient and multiply the Jacobian times a given vector. On the basis of limited experimentation, in order to avoid line searches, the parameter of choice seems to be $\alpha^{(1)}$ used with either $\beta^{(1)}$ or $\beta^{(3)}$ depending on the relative expense of gradient and Jacobian evaluations. Constrained problems are, of course, more expensive to solve, but the conjugate gradient method is an effective approach.

Acknowledgements

Special thanks go to Dr. Paul Concus for his careful reading of the manuscript and his very helpful advice.

References

[1] Axelsson, O.: Solution of linear systems of equations: iterative methods. Sparse Matrix Techniques (Lecture Notes, Vol. 572), pp. 1—11. Berlin-Heidelberg-New York: Springer 1977.

[2] Bartels, R., Daniel, J. W.: A conjugate gradient approach to nonlinear elliptic voundary value problems in irregular regions. Proc. Conf. on Numerical Solution of Differential Equations (Lecture Notes, Vol. 363), pp. 1—11. Berlin-Heidelberg-New York: Springer 1974.

[3] Bertsekas, D.: Partial conjugate gradient methods for a class of optimal control problems. IEEE Trans. Automat. Control *AC-19*, 209—217 (1974).
[4] Broyden, C. G.: Quasi-Newton methods, in: Numerical methods for unconstrained optimization (Murray, W., ed.), pp. 87—106. New York: Academic Press 1972.
[5] Cohen, A. I.: Rate of convergence of several conjugate gradient algorithms. SIAM J. Numer. Anal. *9*, 248—259 (1972).
[6] Concus, P.: Numerical solution of the minimal surface equation. Math. Comp. *21*, 340—350 (1967).
[7] Concus, P.: Numerical solution of the minimal surface equation by block nonlinear successive overrelaxation. Information Processing 68, Proc. IFIP Congress 1968, pp. 153—158. Amsterdam: North-Holland 1969.
[8] Concus, P., Golub, G. H., O'Leary, D. P.: A generalized conjugate gradient method for the numerical solution of elliptic partial differential equations, in: Sparse matrix computations (Bunch, J. R., Rose, D. J., eds.), pp. 309—332. New York: Academic Press 1976.
[9] Concus, P., Golub, G. H., O'Leary, D. P.: Numerical solution of nonlinear elliptic partial differential equations by a generalized conjugate gradient method. Computing *19*, 321—339 (1978).
[10] Daniel, J. W.: The conjugate gradient method for linear and nonlinear operator equations. Ph. D. Thesis, Stanford University, and SIAM J. Numer. Anal. *4*, 10—26 (1967).
[11] Dixon, L. C. W.: Conjugate gradient algorithms: quadratic termination without linear searches. J. Inst. Maths. Applics. *15*, 9—18 (1975).
[12] Douglas, Jesse: A method of numerical solution of the problem of Plateau. Ann. Math. *29*, 180—187 (1928).
[13] Douglas, J., jr., Dupont, T.: Preconditional conjugate gradient iteration applied to Galerkin methods for a mildly nonlinear Dirichlet problem, in: Sparse matrix computations (Bunch, J. R., Rose, D. J., eds.), pp. 333—349. New York: Academic Press 1976.
[14] Eckhardt, U.: On an optimization problem related to minimal surfaces with obstacles. Technical Report, Jülich (1975).
[15] Ehrlich, L. W.: On some experience using matrix splitting and conjugate gradient (abstract). SIAM Review *18*, 801 (1976).
[16] Fletcher, R., Reeves, C. M.: Function minimization by conjugate gradients. Computer J. *7*, 149—154 (1964).
[17] Goldfarb, D.: A conjugate gradient method for nonlinear programming. Princeton University Press, Thesis, 1966.
[18] Greenspan, D.: On approximating extremals of functionals part 1. ICC Bull. *4*, 99—120 (1965).
[19] Hayes, L., Young, D. M., Schleicher, E.: The use of the accelerated SSOR method to solve large linear systems (abstract). SIAM Review *18*, 808 (1976).
[20] Hestenes, M. R.: The conjugate gradient method for solving linear systems. Proc. Symp. in Appl. Math. *6*, 83—102 (1956).
[21] Hestenes, M., Stiefel, E.: Methods of conjugate gradients for solving linear systems. J. Res. Nat. Bur. Stand. *49*, 409—436 (1952).
[22] Klessig, R., Polak, E.: Efficient implementation of the Polak-Ribiere conjugate gradient algorithm. SIAM J. Control *10*, 524—549 (1972).
[23] Lenard, M. L.: Convergence conditions for restarted conjugate gradient methods with inaccurate line searches. Math. Prog. *10*, 32—51 (1976).
[24] McCormick, G. P., Ritter, K.: Alternative proofs of the convergence properties of the conjugate gradient method. J. Opt. Th. Applic. *13*, 497—518 (1974).
[25] Meijerink, J. A., van der Vorst, H. A.: An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. Math. Comp. *31*, 148—162 (1977).
[26] Nazareth, L.: A conjugate direction algorithm without line searches. J. Opt. Th. Applic. *23*, 373—388 (1977).
[27] O'Leary, D. P.: A generalized conjugate gradient algorithm for solving a class of quadratic programming problems. Report STAN-CS-77-638, Stanford University (1977).
[28] Ortega, J. M., Rheinboldt, W. C.: Iterative Solution of Nonlinear Equations in Several Variables. New York: Academic Press 1970.
[29] Polak, E., Ribiere, G.: Note sur la convergence de méthodes de directions conjugées. Rev. Française Informat. Recherche Operationnelle *16-R1*, 35—43 (1969).

[30] Polyak, B. T.: Conjugate gradient method in extremal problems. USSR Comput. Math. and Math. Phys. *9*, 809—821 (1969).
[31] Powell, M. J. D.: Restart procedures for the conjugate gradient method. Math. Prog. *12*, 241—254 (1977).
[32] Reid, J. K.: On the method of conjugate gradients for the solution of large sparse systems of linear equations, in: Large sparse sets of linear equations (Reid, J. K., ed.), pp. 231—254. New York: Academic Press 1971.
[33] Schecter, S.: Relaxation methods for convex problems. SIAM J. Numer. Anal. *5*, 601—612 (1968).
[34] Wang, H. H.: The application of the symmetric SOR and the symmetric SIP methods for the numerical solution of the neutron diffusion equation. Report G 320-3358, IBM Palo Alto Scientific Center (1977).

Dr. Dianne P. O'Leary
Computer Science Department
University of Maryland
College Park, MD 20742, U.S.A.