DISCUSSION PAPER NO. 269

A CLASS OF CONJUGATE GRADIENT ALGORITHMS WITH A TWO-STEP VARIABLE METRIC MEMORY

by

Avinoam Perry

Northwestern University
Graduate School of Management
Evanston, Illinois 60201

January, 1977

Abstract

In this paper we discuss a procedure which replaces the $(n \times n)$ matrix approximating the inverse Hessian of $f(x)$ by four $(n \times 1)$ vectors, two more than required by conjugate gradient methods. The result is a rapidly convergent algorithm which does not require as much computer space as quasi-Newton methods.

Introduction

It is a well-known fact that quasi-Newton algorithms, and especially the BFGS [2,5,8,23] procedure, exhibit more rapid convergence than conjugate gradient algorithms when minimizing the general nonlinear continuous and differentiable function $f(x)$, $x \in R^n$ [11]. However, when n is large, quasi-Newton methods become impractical due to the fact that they require the storing and updating of the $(n \times n)$ matrix approximating the inverse Hessian of $f(x)$. At the same time, conjugate gradient algorithms such as PRCG [9,19], PMCG [17], SCON and SCONS [24], require the storage of only two $(n \times 1)$ vectors containing past information on the function and its derivatives. For this reason, conjugate gradient algorithms replace the quasi-Newton procedures whenever space is scarce due to large-scale problems.

In this paper we propose an interim procedure which does not require as large a storage as the $(n \times n)$ matrix approximating the inverse Hessian of $f(x)$, but, nevertheless, "remembers" two vectors more than required by the above conjugate gradient procedures.

The direct result of such an approach is a method which exhibits more rapid convergence than conjugate gradient procedures.

Conjugate gradient algorithms with memory were recently proposed by Miele and Cantrell [12], M. A. Wolfe [26], Beale [1], and Powell [21]. The basic idea behind these methods was the enforcement of the conjugacy property over all or part of the subspace generated by previous direction vectors. While this property is taken care of automatically by memoryless conjugate gradient procedures when $f(x)$ is quadratic and line search is perfect, it must be enforced explicitly otherwise whenever the conjugacy property is desired.

The conjugacy requirement is effective when the line search part (of the algorithm minimizing $f(x)$) is conducted so as to minimize $h(\alpha) = f(x_k + \alpha d_k)$, $\alpha \geq 0$. However, the high cost associated with perfect line searches is the major reason for replacing the subproblem of minimizing $h(\alpha)$ by the cheaper alternative of finding α_k such that $f(x_k + \alpha_k d_k) < f(x_k)$. Under these circumstances the orthogonality assumption $\nabla f(x_{k+1})' d_k = 0$ no longer holds, and the conjugacy property is not as effective as under perfect line search conditions. Quasi-Newton methods replace the conjugacy requirement by the requirement:

$$(1) \qquad d_k' F d_{k+1} \cong (1/\alpha_k) \cdot q_k' d_{k+1} = -(1/\alpha_k) q_k' S_{k+1} g_{k+1} = -d_k' g_{k+1}$$

It has been demonstrated by Perry [17] that under less than perfect line search conditions conjugate gradient algorithms exhibit better performance if the conjugacy requirement is replaced by the same general requirement (1). Shanno [24] modified Perry's conjugate gradient formula by forcing symmetry and introducing self-scaling, while, at the same time, adopting (1) in place of conjugacy.

The method we propose in this paper differs from previous approaches which incorporate some form of memory into the construction of their new direction vector. While all previous methods seek ways of enforcing the conjugacy property, the new class introduced here replaces this property by requirement (1).

In addition, we also incorporate symmetry and introduce self-scaling.

Our computational experience is very promising and we believe that this form of partial memory will open the door to further studies of space conserving methods for solving the nonlinear function minimization problem.

1. Conjugate Gradient Algorithms With Variable Metric Memory

Conjugate gradient directions such as PRCG and PMCG are constructed by taking a linear combination of the negative gradient at the point x_k and the previous direction vector d_{k-1} which satisfies the equation $x_k = x_{k-1} + \alpha_{k-1} d_{k-1}$ for a given scalar $\alpha_{k-1} > 0$. This approach is demonstrated by:

(2) $\quad d_{k+1} = - g_{k+1} + \beta_k d_k$

where: d_{k+1} is the new direction vector,

$g_{k+1} \equiv \nabla f(x_{k+1})$,

d_k is the previous direction vector,

and β_k is a scalar which is constructed to yield:

(3) $\quad q_k' d_{k+1} = 0$ in PRCG (assuming $d_k' g_{k+1} = 0$),

or

(4) $\quad q_k' d_{k+1} = -p_k' g_{k+1}$ in PMCG ,

where:

$q_k \equiv g_{k+1} - g_k$,

and

$p_k \equiv x_{k+1} - x_k$.

Conjugate gradient algorithms with variable metric memory can be constructed by taking a linear combination of the direction vector d_k and the negative gradient g_{k+1} defined over a different metric represented by the $(n \times n)$ positive definite matrix S.

This approach is demonstrated by (5).

(5) $\quad d_{k+1} = -S g_{k+1} + \beta_k d_k$.

Again, β_k can be constructed to yield (3) such that

(6) $$\beta_k = \frac{q_k' S g_{k+1}}{q_k' d_k}$$

and (5) becomes:

(7) $$d_{k+1} = -Sg_{k+1} + \frac{q_k' S g_{k+1}}{q_k' d_k} \cdot d_k .$$

(7) can be written as:

(7a) $$d_{k+1} = -\left[S - \frac{p_k q_k' S}{q_k' p_k} \right] g_{k+1} = -Dg_{k+1} .$$

Since D is not positive definite, then a method based on (7a) may become unstable even if S is positive definite (see [24] for explanation). To correct for this deficiency the matrix D can be changed to S^* where

(8) $$S^* = D - \frac{S q_k p_k'}{p_k' q_k} + \left(1 + \frac{q_k' S q_k}{p_k' q_k}\right) \frac{p_k p_k'}{p_k' q_k} ,$$

and

(9) $$d_{k+1} = -S^* g_{k+1} .$$

Note that (9) is identical to (7) whenever line search is perfect. If S^* is used in place of S in constructing the next direction vector, then (8)-(9) is exactly the BFGS [2,5,8,23] method.

Assuming $d_k = -Sg_k$, then conjugate gradient algorithms with variable metric memory can also be constructed by taking a linear combination of the following type:

(10) $$d_{k+1} = -Sg_{k+1} + \gamma_k S q_k$$

where:

(11) $$\gamma_k = \frac{q_k' S g_{k+1}}{q_k' S q_k} .$$

By letting

(12) $Sq_k = d_k + Sg_{k+1}$

(10) becomes

(13) $d_{k+1} = -Sg_{k+1} + \gamma_k(d_k + Sg_{k+1}) = -(1-\gamma_k)Sg_{k+1} + \gamma_k d_k$

and by letting $\beta_k = \frac{\gamma_k}{1-\gamma_k}$, (13) becomes identical to (7). (10) can be written as:

(14) $$d_{k+1} = -\left[S - \frac{Sq_k q'_k S}{q'_k S q_k} \right] g_{k+1} = -Dg_{k+1} \; .$$

Since D is not positive definite, it can be corrected by defining

(15) $$S^* = D + \frac{p_k p'_k}{p'_k q_k}$$

and applying (9) in place of (14).

Note that (10) is identical to (15) whenever line search is perfect. If S^* is used in place of S in constructing the succeeding direction vector, then (15) is exactly the DFP [3,6] method.

Quasi-Newton methods such as the DFP and the BFGS procedures are special cases of conjugate gradient methods with full variable metric memory. However, if full memory is replaced by partial memory, which is represented by linear combinations of two vectors rather than the $(n \times n)$ matrix S, then problems with large number of variables can enjoy more rapid convergence than if solved by the memoryless conjugate gradient methods.

The procedure we propose is summarized below:

Consider the BFGS formula:

(16) $$d_{k+1} = -\left[S_k - \frac{p_k q'_k S_k}{p'_k q_k} - \frac{S_k q_k p'_k}{p'_k q_k} + \left(1 + \frac{q'_k S_k q_k}{p'_k q_k}\right) \frac{p_k p'_k}{p'_k q_k} \right] g_{k+1}$$

Let:

(17) $$y_k = S_k q_k$$

then (16) can be written as:

(18) $$d_{k+1} = -\left[S_k - \frac{p_k y_k'}{p_k' q_k} - \frac{y_k p_k'}{p_k' q_k} + \left(1 + \frac{q_k' y_k}{p_k' q_k}\right) \frac{p_k p_k'}{p_k' q_k} \right] g_{k+1}$$

$$= -S_k g_{k+1} + \frac{y_k' g_{k+1}}{p_k' q_k} \cdot p_k + \frac{p_k' g_{k+1}}{p_k' q_k} \cdot y_k - \left(1 + \frac{q_k' y_k}{p_k' q_k}\right) \frac{p_k' g_{k+1}}{p_k' q_k} \cdot p_k$$

Upon letting

(19) $$S_k g_{k+1} = y_k - d_k$$

(18) becomes

(20) $$d_{k+1} = \left(-1 + \frac{d_k' g_{k+1}}{d_k' q_k}\right) y_k - \left[- \frac{y_k' g_{k+1}}{d_k' q_k} + \frac{q_k' y_k \, d_k' g_{k+1}}{(d_k' q_k)^2} + \frac{\alpha_k d_k' g_{k+1}}{d_k' q_k} - 1 \right] d_k$$

and upon replacing (17) by:

(21) $$y_k = \begin{cases} q_k - \dfrac{p_{k-1}' q_k}{p_{k-1}' q_{k-1}} \cdot q_{k-1} + \left[\left(1 + \dfrac{q_{k-1}' q_{k-1}}{p_{k-1}' q_{k-1}}\right) \dfrac{p_{k-1}' q_k}{p_{k-1}' q_{k-1}} - \dfrac{q_{k-1}' q_k}{p_{k-1}' q_{k-1}} \right] p_{k-1}, & \text{if } k \geq 2 \\ q_k & \text{, if } k = 1 \end{cases}$$

We can construct the following algorithm denoted here as TSVM.

```
Step 0:  set $d_0 = -g_0$
Step 1:  find $\alpha_k$ satisfying $f(x_k + \alpha_k d_k) < f(x_k)$
Step 2:  set $x_{k+1} = x_k + \alpha_k d_k$
Step 3:  let $g_{k+1} = \nabla f(x_{k+1})$, $p_k = x_{k+1} - x_k$, $q_k = g_{k+1} - g_k$
Step 4:  if $|g_{k+1}| \leq \varepsilon$ stop.  Otherwise go to 5.
Step 5:  define $y_k$ as in (21)
Step 6:  define $d_{k+1}$ as in (20)
Step 7:  let $p_{k-1} \equiv p_k$, $q_{k-1} \equiv q_k$, $g_k \equiv g_{k+1}$, $x_k \equiv x_{k+1}$, $d_k \equiv d_{k+1}$ and go to 1.
```

It should be noted that there exists a full class of two-step variable metric memory algorithms. An important member of this class, which is denoted here as TSVM2 is a variation of DFP where the full memory matrix S_k is replaced by the two step memory y_k such that the direction vector is constructed by the following equation:

$$(22) \quad d_{k+1} = \left[\frac{y_k' g_{k+1}}{y_k' q_k} - 1 \right] y_k + \left[1 - \frac{\alpha_k d_k' g_{k+1}}{d_k' q_k} \right] d_k \quad .$$

2. Self-Scaling Conjugate Gradient With a Two-Step Variable Metric Memory

Self-scaling of variable metric algorithms which improves overall numerical performance of these methods were first introduced by Oren [13], and Oren and Luenberger [14]. These procedures were then refined by Oren and Spedicato [15] and by Shanno and Phua [25]. In a recent paper, Shanno [24] applied the self-scaling procedure to a new conjugate gradient algorithm (SCONS) and his reported computational experience looks very promising.

In this paper we use a procedure similar to the one suggested by Shanno [24], but modify it to fit the two-step variable metric memory algorithm.

For reasons which are presented and discussed in Luenberger [10], self-scaling is a tool which improves local convergence properties of quasi-Newton algorithms. The main objective of the self-scaling procedure is to spread the eigen values of S_kF_k (where S_k is the matrix approximating the inverse Hessian of $f(x_k)$, and where F_k is the actual Hessian of $f(x_k)$) around unity. This objective can be achieved by scaling S_k such that the self-scaling BFGS procedure becomes:

$$(23) \qquad S_{k+1} = \gamma_k \left[S_k - \frac{p_k q_k' S_k}{q_k' p_k} - \frac{S_k q_k p_k'}{q_k' p_k} \right] + \left(1 + \gamma_k \frac{q_k' S_k q_k}{q_k' p_k} \right) \cdot \frac{p_k p_k'}{p_k' q_k}$$

where:

$$(24) \qquad \gamma_k = \frac{p_k' q_k}{q_k' S_k q_k} .$$

Shanno and Phua [25] modified the self-scaling procedure by letting

$$(25) \qquad \gamma_k = \begin{cases} 1 & \text{for } k \geq 1 \\ \dfrac{p_k' q_k}{q_k' S_k q_k} & \text{for } k = 0 \end{cases}$$

and denoted this self-scaling procedure as BFGS18.

Shanno [24] applied the same principles to the memoryless BFGS in which $S_k = S_0 = I$, and applied the **scalar**

$$(26) \qquad \gamma_k = \frac{p_k' q_k}{q_k' q_k} .$$

Applying the same principles and using the same reasonings as in [24] the scaler γ_k which we apply to the two step variable metric memory algorithm is

$$(27) \qquad \gamma_k = \frac{p_k' q_k}{q_k' S_k q_k} = \frac{p_k' q_k}{q_k' y_k}$$

and the self-scaling TSVMS direction vector becomes:

$$(28) \qquad d_{k+1} = (-1 + \frac{d_k' g_{k+1}}{d_k' q_k}) \ \gamma_k y_k - \left[\frac{-\gamma_k y_k' g_{k+1}}{d_k' q_k} + \frac{\gamma_k q_k' y_k \cdot d_k' g_{k+1}}{(d_k' q_k)^2} + \frac{d_k' g_{k+1}}{d_k' q_k} - \gamma_k \right] . d_k$$

where y_k is defined in (21).

Computational Experience

We compared the performance of TSVM and TSVMS (TSVM with self-scaling) to BFGS [2,5,8,23], BFGS18 [25], SCON, SCONS [24], and SCCG [18]. Experiments involved six well known test functions which are denoted here as functions 1 through 6 respectively (see appendix for a detailed description of each function and its source). All problems were solved by each one of the algorithms above under two different line search accuracy measures. The line search technique applied is the well known quadratic interpolation method [7]. In order to ensure successful implementation of the line search procedure a unimodal region was secured before the first interpolation was performed. Line search accuracy was measured by the index:

$$\delta = \frac{|p'_{k+1}g_{k+2}|}{|p'_{k+1}g_{k+1}|} \tag{29}$$

Performance of a given algorithm was measured by two separate data; total number of stages and total number of function evaluations (in parentheses), respectively. We use the term "stage" to define the step carrying a point x_k along a direction d_k to new point $x_{k+1} = x_k + \alpha_k d_k$.

The statistics "total number of function evaluations" includes the number of gradient evaluations multiplied by n (if one is interested in number of function evaluations not including gradient evaluations, the total number of stages multiplied by n (or 2n in the case of SCCG [18]) should be subtracted from the total number of function evaluations). Each time a direction vector pointed upwards, rather than downwards, it was replaced by the direction of steepest descent.

The stopping rule applied throughout was

$$|\nabla f(x^*)| \leq 10^{-5} \tag{30}$$

All computer programs were coded in APL using interactive mode [16] and were run on the CDC 6400 computer at Northwestern University.

In the following tables we present our computational results under two measures of line search accuracy. These measures are denoted as mode 1 ($\delta \leq .1$) and mode 2 ($\delta \leq .001$) where δ is defined as in (29). The one dimensional search was terminated whenever the above constraint became satisfied. Our computational study is divided into two parts. In the first part no restarts were initiated, and in the second part all algorithms were restarted after a multiple of (n+1) stages (where n is the dimension of the function in question).

Concluding Remarks

As reflected by our computational results the two step variable metric memory algorithm performed better than the memoryless conjugate gradient algorithm, but was somewhat inferior to the BFGS. This conclusion is consistent whether one does or does not apply self scaling. Self scaling tends to improve speed of convergence in most cases while occasional restarts are useful only in the case of memoryless conjugate gradient.

Based on these findings we recommend the following strategy for the function minimization problem:

(a) Whenever sufficient computer space is available, apply BFGS18 [25] with no restarts.

(b) If the function to be minimized is of a relatively large dimension so that BFGS18 is infeasible, one should apply TSVMS with no restarts.

(c) If application of TSVMS becomes infeasible due to the fact that n is extremely large, then either SCONS [24] or SCCG [18] should be used.

(d) The decision on whether SCONS or SCCG should be used depends on the amount of effort required in evaluating $\nabla f(x)$. Since SCCG requires one extra gradient evaluation per iteration, one should prefer SCONS to SCCG whenever the evaluation of $\nabla f(x)$ is a significant time consuming task. However, since SCCG usually solves a given problem in less iterations than SCONS, it follows that whenever the evaluation of $\nabla f(x)$ comes cheaply SCCG should be preferred to SCONS.

Our computational study indicates that TSVMS is superior to both SCONS and SCCG. Since the difference in storage requirement between TSVMS and SCONS or SCCG is only two $(n \times 1)$ vectors, there is a reason to believe that in most real applications TSVMS will be feasible as long as SCONS or SCCG are.

Function 1

x_0 = -1.2, 1

	No Restarts		Restarts	
Algorithm	Mode 1	Mode 2	Mode 1	Mode 2
BFGS	25(214)	18(250)	33(359)	29(443)
TSVM	25(214)	18(250)	33(330)	29(413)
SCON	23(204)	18(245)	27(284)	29(443)
BFGS18	27(238)	24(293)	42(410)	30(409)
TSVMS	25(222)	22(275)	47(457)	30(406)
SCONS	26(224)	23(281)	45(431)	30(409)
SCCG	16(173)	13(219)	14(171)	17(285)

Function 2

x_0 = -3, -1, -3, -1

	No Restarts		Restarts	
Algorithm	Mode 1	Mode 2	Mode 1	Mode 2
BFGS	38(373)	39(539)	69(793)	54(769)
TSVM	38(423)	31(407)	28(332)	45(604)
SCON	154(1439)	136(1493)	134(1625)	65(898)
BFGS18	20(231)	24(312)	25(300)	33(453)
TSVMS	33(363)	35(455)	46(570)	35(487)
SCONS	136(1460)	132(1457)	43(527)	42(566)
SCCG	58(769)	58(811)	15(232)	19(337)

Function 3

x_0 = 3, -1, 0, 1

	No Restarts		Restarts	
Algorithm	Mode 1	Mode 2	Mode 1	Mode 2
BFGS	19(192)	18(257)	14(170)	22(340)
TSVM	12(131)	41(562)	12(128)	65(944)
SCON	212(2278)	218(2373)	53(639)	65(934)
BFGS18	27(282)	27(344)	54(654)	39(567)
TSVMS	36(353)	35(455)	50(634)	30(430)
SCONS	136(1407)	76(923)	58(739)	50(726)
SCCG	56(706)	53(818)	43(698)	41(784)

Function 4

x_0 = [-2, -2, -2, -2, -2, -2, -2, -2, -2, -2]

	No Restarts		Restarts	
Algorithm	Mode 1	Mode 2	Mode 1	Mode 2
BFGS	32(517)	31(577)	44(687)	42(743)
TSVM	37(602)	29(508)	42(681)	40(714)
SCON	44(692)	29(511)	43(693)	39(698)
BFGS18	26(412)	25(443)	32(519)	29(525)
TSVMS	35(532)	28(491)	33(531)	30(542)
SCONS	41(650)	32(568)	39(644)	32(569)
SCCG	27(691)	30(802)	26(662)	24(692)

Function 5

x_0 = 1, 0, 0, 0

	No Restarts		Restarts	
Algorithm	Mode 1	Mode 2	Mode 1	Mode 2
BFGS	23(221)	23(347)	45(511)	10(158)
TSVM	24(236)	34(530)	54(583)	44(645)
SCON	33(342)	15(239)	31(339)	21(338)
BFGS18	25(251)	21(307)	21(204)	16(231)
TSVMS	35(532)	28(491)	27(288)	26(307)
SCONS	45(450)	40(482)	32(327)	30(341)
SCCG	13(174)	13(258)	12(158)	11(251)

Function 6

x_0 = [-2, -2, -2, -2, -2, -2, -2, -2, -2, -2]

	No Restarts		Restarts	
Algorithm	Mode 1	Mode 2	Mode 1	Mode 2
BFGS	43(688)	42(729)	23(368)	23(401)
TSVM	19(292)	17(304)	16([illegible]37)	14(291)
SCON	11(172)	13(248)	11(172)	10(183)
BFGS18	30(468)	28(523)	25(388)	27(421)
TSVMS	15(234)	12(231)	15(238)	13(235)
SCONS	16(250)	14(263)	12(191)	12(212)
SCCG	15(369)	10(357)	14(338)	11(340)

Appendix

The following functions were used in our study.

1. $f = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$ [22]

2. $f = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2)^2 + (1 - x_3)^2$ [27]

3. $f = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4$ [20]

4. $f = (1 - x_1)^2 + (1 - x_{10})^2 + \sum_{i=1}^{9} (x_i^2 - x_{i+1})^2$ [4]

5. $f = (\exp(x_1) - x_2)^4 + 100(x_2 - x_3)^6 + \{\tan^{-1}(x_3 - x_4)\}^4 + x_1^8$ [12]

6. $f = (\sum_{i=1}^{10} ix_i^2)^2$ [25]

References

[1] Beale, E.M.L., "A Derivation of Conjugate Gradients" in F. A. Lootsma, ed., "Numerical Methods for Nonlinear Optimization", Academic Press, London, 1972, pp. 39-43.

[2] Bryoden, C.G., "The Convergence of a Class of Double Rank Algorithms, Parts I and II," J. Inst. Math. Appl., 7, 1971, pp. 76-90, pp. 222-236.

[3] Davidon, W.C., "Variable Metric Method for Minimization," R&D Report ANL-5990, U.S. Atomic Energy Commission, Argonne National Laboratories, 1959.

[4] Dixon, L.C.W., J. Inst. Math. Appl., 11, 1973, pp. 317-328.

[5] Fletcher, R., "A New Approach to Variable Metric Algorithms," Com. J., 13, 1970, pp. 317-322.

[6] Fletcher, R. and M.J.D. Powell, "A Rapidly Convergent Descent Method for Minimization," Comp. J., 6, 1963, pp. 163-168.

[7] Fox, R.L., "Optimization Methods for Engineering Design," Addison-Wesley, 1973, pp. 51-55.

[8] Goldfarb, D., "A Family of Variable Metric Methods Derived by Variational Means," Math. Comp. 24, 1970, pp. 23-26.

[9] Klessig, R. and E. Polak, "Efficient Implementation of the Polak-Ribiere Conjugate Gradient Algorithm," Siam J. Control, 10, 1972, pp. 524-549.

[10] Luenberger, D.G., "Introduction to Linear and Nonlinear Programming," Addison-Wesley Publishing Co., 1973, pp. 201-204.

[11] McCormick, G.P. and K. Ritter, "Methods of Conjugate Directions Versus Quasi-Newton Methods," Math. Prog., 3, 1972, pp. 101-111.

[12] Miele, A. and J.W. Cantrell, "A Memory Gradient Method for Unconstrained Minimization," J. of Opt. Theory Appl., 3, 1969, pp. 459-470.

[13] Oren, S.S., "Self Scaling Variable Metric Algorithms, Part II," Management Science, Vol. 20, No. 5, 1974, pp. 863-874.

[14] Oren, S.S. and D.G. Luenberger, "Self Scaling Variable Metric Algorithms Part I," Management Science, Vol. 20, No. 5, 1974, pp. 845-862.

[15] Oren, S.S. and E. Spedicato, "Optimal Conditioning of Self Scaling Variable Metric Algorithms," Math. Prog., Vol. 10, No. 1, 1976, pp. 70-90.

[16] Perry, Avinoam, "UCNLP-An Iteractive Package of Programs for Unconstrained Nonlinear Optimization Purposes," A Working Paper, Nov. 1975, Revised May 1976.

[17] Perry, Avinoam, "A Modified Conjugate Gradient Algorithm," forthcoming in Operations Research.

[18] Perry, Avinoam, "A Self Correcting Conjugate Gradient Algorithm," Discussion Paper No. 228, The Center for Mathematical Studies in Economics and Management Science, Northwestern University, Evanston, IL 1976.

[19] Polak, E. and G. Ribiere, "Note Sur La Convergence de Methodes de Directions Conugess," Revue Francaise Inf. Rech. Oper., 16 RI 1969, pp. 35-43.

[20] Powell, M.J.D., "An Iterative Method for Finding Stationary Values of a Function with Several Variables Without Calculating Derivatives," Computer J., 7(4), 1964, pp. 155-162.

[21] Powell, M.J.D., "Restarts Procedures for the Conjugate Gradient Method," Harwell Report C.S.S., 24, November 1975.

[22] Rosenbrock, H.H., "Automatic Method for Finding the Greatest or the Least Value of a Function," Computer Journal, 3(3), 1960, pp. 175-184.

[23] Shanno, D.F., "Conditioning of Quasi-Newton Methods for Function Minimization," Math. Comp., 24, 1970, pp. 647-656.

[24] Shanno, D.F., "Conjugate Gradient Methods With Inexact Searches," Working Paper, Department of MIS, College of Business and Public Administration, University of Arizona, Tucson, AZ, 1976.

[25] Shanno, D.F. and K.M. Phua, "Matrix Conditioning and Nonlinear Optimization," Working Paper, Dept. of MIS, College of Business and Public Administration, University of Arizona, Tucson, AZ, 1976.

[26] Wolfe, M.A., "A Quasi-Newton Method with Memory for Unconstrained Function Minimization," J. Inst. Math. Appl., 15, 1975, pp. 85-94.

[27] Wood, C.F., Westinghouse Research Labs, 1968.