T-3141

A NEW EXPANDING SUBSPACE ALGORITHM FOR NONLINEAR OPTIMIZATION

ARTHUR LAKES LIBRARY
COLORADO SCHOOL of MINES
GOLDEN, COLORADO 80401

by
D. J. MacMillan

ProQuest Number: 10782757

All rights reserved

INFORMATION TO ALL USERS
The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript
and there are missing pages, these will be noted. Also, if material had to be removed,
a note will indicate the deletion.

ProQuest 10782757

Published by ProQuest LLC (2018). Copyright of the Dissertation is held by the Author.

All rights reserved.
This work is protected against unauthorized copying under Title 17, United States Code
Microform Edition © ProQuest LLC.

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

T-3141

A dissertation submitted to the Faculty and the Board of Trustees of the Colorado School of Mines in partial fulfillment of the requirements for the degree of Master of Science (Mathematics).

Golden, Colorado

Date November 22, 1985

Signed: Daniel J. MacMillan

Daniel J. MacMillan

Approved: Robert G. Underwood

Dr. Robert G. Underwood
Thesis Advisor

Golden, Colorado

Date Nov. 22, 1985

Dr. Ardel J. Boes
Head of Department of
Mathematics

ARTHUR LAKES LIBRARY
COLORADO SCHOOL of MINES
GOLDEN, COLORADO 80401

ABSTRACT

A nonlinear optimization algorithm, BANANA, is presented. This algorithm has been designed to solve unconstrained minimization problems where the computer time required for a function evaluation is large, the number of variables is small, and the eigenvalues of the objective function's Hessian are illconditioned. The motivation for developing this algorithm is the estimation of parameters in an oil reservoir model from measured data using a least squares approach. The implementation of this problem uses twelve to fifteen variables, one function evaluation takes an average of 45 seconds on a Burroughs B7900 computer at the rate of 0.4 MFLOPS; and the ratio of the largest to smallest eigenvalue is 8.0×10^{11}.

The BANANA algorithm is compared to other frequently used algorithms using several standard test problems as well as the oil reservoir problem. BANANA was found to use significantly fewer function evaluations; and, it was the only algorithm to successfully compute estimates of the parameters in the oil reservoir problem when using field data.

TABLE OF CONTENTS

Page

APPENDICES

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would line to acknowledge Marathon Oil Company for its generous support throughout my pursuit of this degree. I would also like to express my appreciation for the use of Marathon's computing facilities in developing and testing the nonlinear optimization algorithm presented here. Finally, I wish to thank Lois Fitzpatrick whose efforts in typing this manuscript are much appreciated.

INTRODUCTION

The nonlinear optimization algorithm presented here, BANANA, has been designed to solve unconstrained minimization problems where the computer time required for a function evaluation is large, the number of variables is small and the eigenvalues of the objective function's Hessian are illconditioned. The specific problem behind the development of the algorithm is the estimation of parameters in an oil reservoir model from measured data using a least squares approach. In our implementation of this problem, twelve to fifteen variables are used, one function evaluation takes an average of 45 seconds on the Burroughs B7900 computer at the rate of 0.4 MFLOPS; and the ratio of the largest to smallest eigenvalue is 8.0×10^{11}. Therefore, our criterion for comparing the performance of different algorithms is the number of function evaluations required to find the optimal solution.

We ran BANANA, modified Newton [1] and International Mathematical and Statistical Libraries, Inc. (IMSL) code for conjugate gradient [2] and quasi-Newton [3] on several standard test problems plus a few of our own design. We

also tried the SPIRAL [4] algorithm but it diverged for the oil reservoir model. For the problems that were essentially quadratic, BANANA took a comparable number of function calls. For the others, BANANA took 30 to 70 percent fewer function evaluations than the next best algorithm. These algorithms, along with the IMSL Marquardt [5-8] code were also run on the oil reservoir problem using real and computer generated data. The modified Newton and Marquardt algorithms managed to out perform BANANA in a couple of isolated cases. However, BANANA was the only algorithm that found the minimum of all these test problems, and BANANA was also the only algorithm that found the minimum for the real data test case.

The BANANA algorithm is illustrated in two dimensions in Figure 1. The solid banana shaped curve is a level curve of an illconditioned objective function. The vectors e_1 and e_2 are eigenvectors of the objective function's Hessian where e_1 corresponds to the largest eigenvalue and e_2 to the smallest. The first direction searched is e_1 evaluated at point A. After this search is completed, minimization takes place over the entire space by moving from point B to the minimum at C along a path that approximates the dashed curve

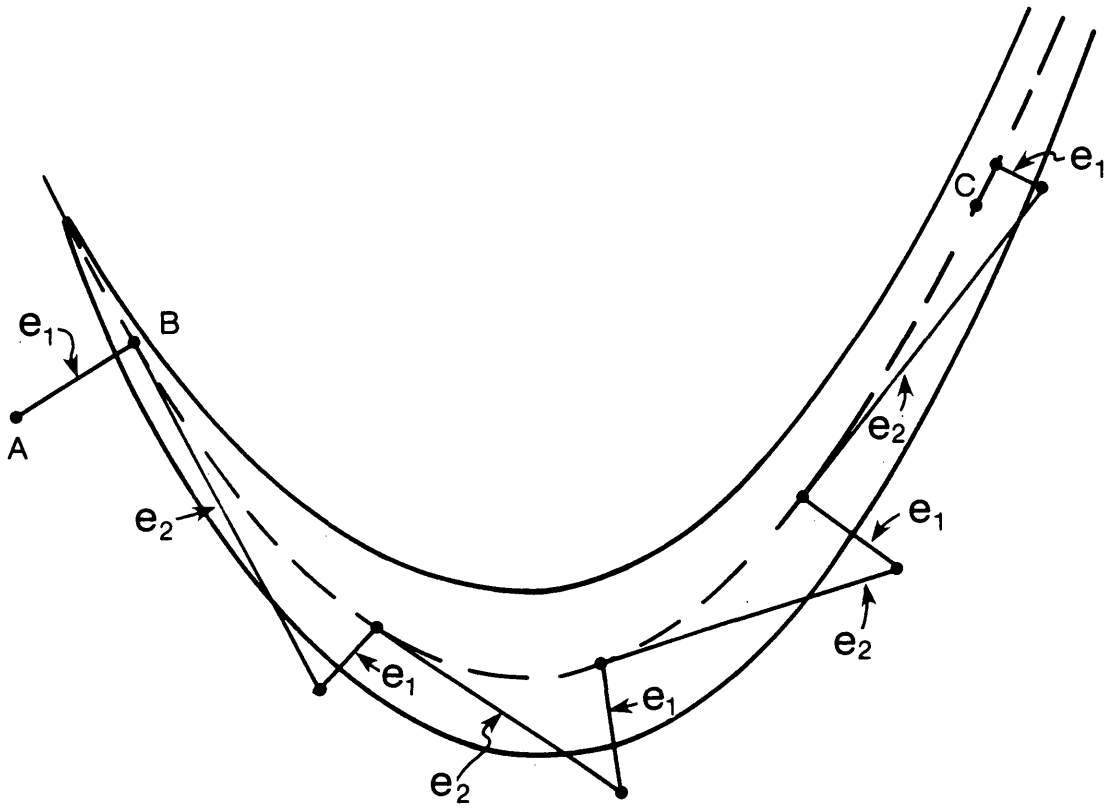

Figure 1. Two Dimensional Illustration of the BANANA Alogorithm.

in the figure. To do this, BANANA first steps along an e2 eigenvector direction until a point with an increase in the objective function is obtained. We do not perform an accurate line search here. Instead, an e_1 eigenvector direction is searched starting at this point to find a new point on the dashed curve. This process is repeated (as illustrated by the zigzag curve of Figure 1) until the algorithm is in the neighborhood of the minimum. Then, the algorithm switches to Newton's method to complete the minimization process. We shall refer to this sequence of searches as acceleration along the valley. Acceleration along the valley takes advantage of illconditioning since steps along the e_2 direction do not wander too far from the valley floor represented by the dashed curve BC.

If the dashed curve in Figure 1 were linear, as would be the case for a quadratic objective function, then BANANA searches the two eigenvector directions which in this case become conjugate directions. So, for quadratic objective functions, BANANA is a conjugate direction method. For objective functions where the dashed curve is not linear, BANANA modifies the conjugate direction method by searching the curved path as indicated in Figure 1. Thus, BANANA is a

modified conjugate direction method that searches a sequence of "conjugate" curves to obtain the minimum of the objective function.

When BANANA is generalized to n dimensions, a stage of the algorithm consists of a search in a "subspace", a linear variety [9], followed by a search along a "conjugate" curve. The linear variety is then expanded and the process is repeated. These linear varieties are translated subspaces, S, having bases of eigenvectors of the objective function's Hessian. Each succeeding linear variety includes more eigenvectors in S. Observe that if the largest eigenvalue for a positive definite quadratic objective function is several orders of magnitude larger than the other eigenvalues then the corresponding eigenvector points in essentially the negative gradient direction. In order to handle this illconditioned situation, eigenvectors with eigenvalues similar in magnitude are grouped together. Thus, we include one more group of eigenvectors in each succeeding linear variety in the order of dereasing eigenvalues. Note that each succeeding stage of the algorithm starts out in essentially the negative gradient direction if the eigenvalues remain illconditioned. The reason for this is that the

gradient is perpendicular to the eigenvectors in the previous linear variety due to the objective function having been minimized over that linear variety.

DEFINITIONS AND PRELIMINARY DISCUSSION

Let R^n represent the vector space of n-tuples with Euclidian norm. Assume that the objective function, f(x), maps R^n into R and has continuous first and second partial derivatives through out R^n. We denote the gradient of f(x) by g(x) and the Hessian by H(x). We also denote the matrix of eigenvectors of the Hessian by $E(x) = \{e_1(x), \ldots, e_n(x)\}$ and the diagonal matrix of eigenvalues, $\lambda_i(x)$, by $\Lambda(x)$. Eigenvectors and eigenvalues are always assumed to be in the order of the largest to smallest eigenvalue. Let I be the index set $I = \{1, \ldots, n\}$ and P be a subset of I.

Definition 1:

We define the linear variety, $S_P(x)$, by $S_P(x) = x + \text{span}\{e_i(x), i \varepsilon P\}$ and refer to it as a subvariety.

In our algorithm, Newton's method [1] is modified by projecting it onto a subvariety. This direction is then searched to minimize the objective function within the subvariety.

Newton's algorithm is

$$x_{k+1} = x_k - H^{-1}(x_k)g(x_k) \tag{1}$$

which for the eigenvector basis becomes

$$\tilde{x}_{k+1} = \tilde{x}_k - \Lambda^{-1}(x_k)\tilde{g}(x_k) \tag{2}$$

where

$$\tilde{x}_k = E(x_k)x_k \tag{3}$$

and

$$\tilde{g}(x_k) = E(x_k)g(x_k) \quad . \tag{4}$$

Let

$$\tilde{d}(x_k) = - \Lambda^{-1}(x_k)\tilde{g}(x_k) \quad . \tag{5}$$

Then, each individual element of $\tilde{d}(x_k)$ is given by

$$\tilde{d}_i(x_k) = - \frac{\tilde{g}_i(x_k)}{\lambda_i(x_k)} \quad . \tag{6}$$

Now, for a general subvariety, $S_P(x_k)$, we define the new projected direction vector $\tilde{d}_P(x_k)$ that has $\tilde{d}_i(x_k)$ for its ith component if $i \varepsilon P$ or zero if $i \not\varepsilon P$. Thus, we have

$$d_P(x_k) = E^T(x_k)\tilde{d}_P(x_k) \quad . \tag{7}$$

This gives a line $[x_k + \alpha d_P(x_k)]$ that is contained in the subvariety $S_P(x_k)$. This line is searched by BANANA in order to find the value of α that minimizes $f(x_k + \alpha d_P(x_k))$. Appendix A describes how the step size is chosen for these searches. To handle the possibility of negative eigenvalues and assure that $f(x_k + \alpha dp(x_k))$ decreases as α increases from zero, we

use the absolute value of $\lambda_i(x_k)$ in (6).

Our criterion for convergence on a subvariety, $S_p(x_k)$, is

$$\left|\tilde{d}_i(x_k)\right| < \tau \text{ for all } i \varepsilon P. \qquad (8)$$

We found that $\tau = 0.01$ works well. It may be possible to improve the performance of the algorithm by using a different τ or convergence criterion.

A stage of the algorithm, denoted by the subscript ℓ, consists of two parts as illustrated in Figure 2. First, an accurate minimization is done on a subvariety which we call a hypercross section. This takes the algorithm from point A to point B. Second, we accelerate along the dashed curve in the figure until we loosely approximate the minimum on this curve which is at point C. We call this curve a hypervalley. At the end of the stage, the hypercross section is expanded to include the eigenvector directions used to define the hypervalley. Hypercross sections and hypervalleys are chosen using the index sets defined recursively below.

Definition 2:

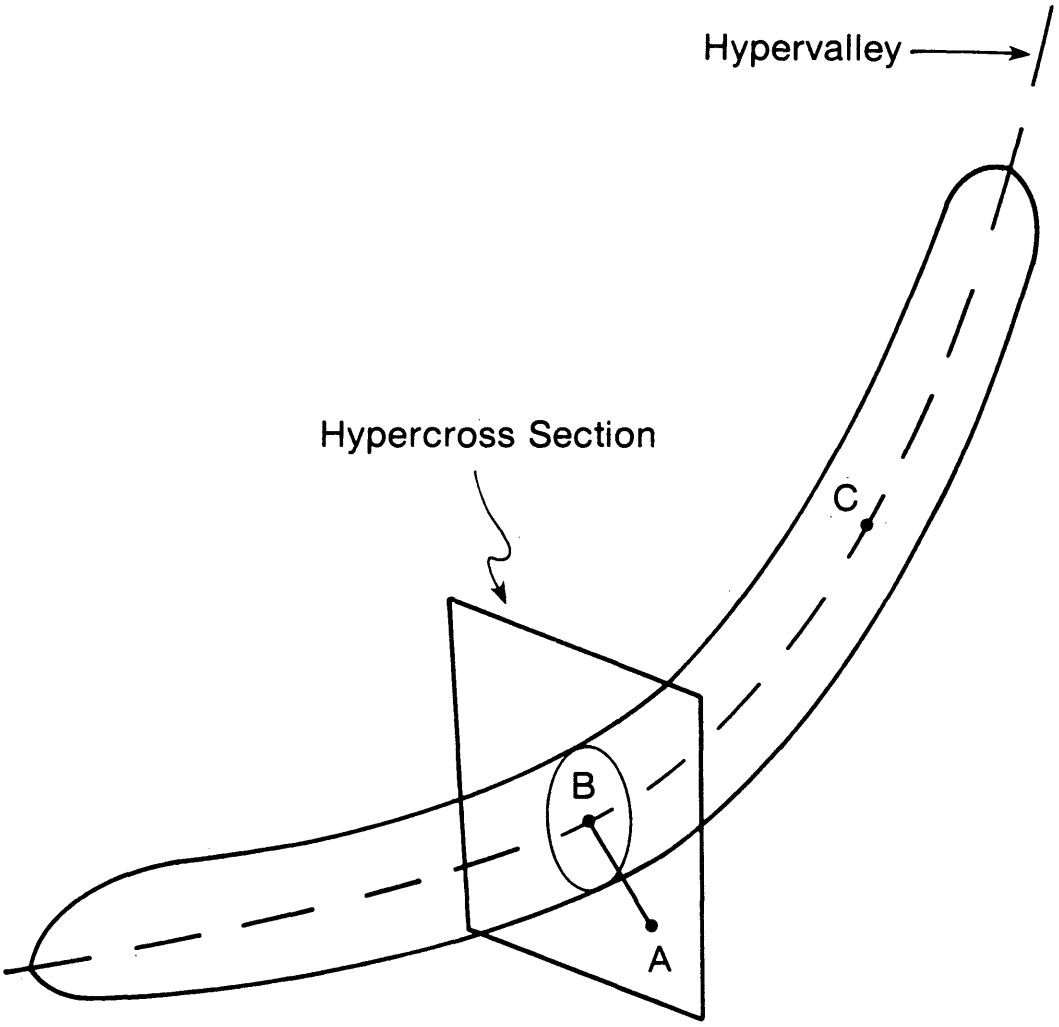

Figure 2. Three Dimensional Illustration of a Stage of the BANANA Alogrithm.

$$C_1 = \{i \varepsilon I, \lambda_i(x) \geq \gamma \lambda_1(x)\}, \text{ where } C_1 \text{ contains } m_1 \text{ elements.} \quad (9)$$

$$V_1 = \{i \varepsilon C_1^c, \lambda_i(x) \geq \gamma \lambda_{m_1+1}(x)\} \quad (10)$$

$$C_\ell = C_{\ell-1} \cup V_{\ell-1}, \text{ where } C_\ell \text{ contains } m_\ell \text{ elements.} \quad (11)$$

$$V_\ell = \{i \varepsilon C_\ell^c, \lambda_i(x) \geq \gamma \lambda_{m_\ell+1}(x)\} \quad (12)$$

The superscript c indicates the complement. The argument, x, is left off the sets C_ℓ and V_ℓ because these sets are defined at the start of each stage of the algorithm and remain constant during that stage. The parameter, γ, is used to group eigenvectors with eigenvalues of similar magnitude together where $0 \leq \gamma \leq 1$. If $\gamma = 1$, then only eigenvectors with the same eigenvalue are grouped together. If $\gamma = 0$, then all the eigenvectors are grouped together. We have found that $\gamma = 1/2$ works well. It may be possible to improve the performance of the algorithm by using a different value for γ or by using a different grouping criterion. The subvariety $S_{C_\ell}(x)$ is the hypercross section for stage ℓ.

Definition 3:

The hypervalley for stage ℓ, $x_\ell(t)$, is defined by the differential equation

$$\frac{dx_\ell(t)}{dt} = d_{V_\ell}(x_\ell(t)) \tag{13}$$

where $x_\ell(0)$ is the point that gave the minimum of the objective function for the hypercross section $S_{C_\ell}(x)$ and d_{V_ℓ} is given by (7).

If the hypervalley is linear, the acceleration procedure illustrated in Figure 1 is not necessary and we can save function calls by searching the direction $d_{V_\ell}(x_\ell(t))$.

Definition 4:

A hypervalley is defined as linear if $d_{V_\ell}(x_\ell(t))$ is constant for all t in (13).

However, this definition is not very useful for implementation in our algorithm. Therefore, since $S_{C_\ell}(x_\ell(t))$ satisfies the convergence criterion given by (8) for all t, we assume that the hypervalley is sufficiently linear if $S_{C_\ell}[x_\ell(0) + \alpha d_{V_\ell}(x_\ell(0))]$ satisfies this convergence criterion for some value of α such that $f[x_\ell(0)] < f[(x_\ell(0) + \alpha d_{V_\ell}(x_\ell(0))]$.

The hypercross section is expanded when the algorithm is near the minimum on a hypervalley. Therefore, we need to be able to determine when we are near this minimum. This is done by testing two consecutive $d_{V_\ell}(x_\ell(t))$ directions as illustrated in Figure 3. When we are away from the minimum, these vectors will point in the same direction as d_1 and d_2 in the figure. When we are near the minimum, these vectors point in opposite directions as d_2 and d_3 in the figure. Assume that $f(x_\ell(t))$ is strictly convex for all t in (12). Then, there exists t^* such that $f(x_\ell(t^*)) \leq f(x_\ell(t))$ for all t. Thus, the minimum of $f(x_\ell(t))$ is at $x_\ell(t^*)$. If $x_\ell(t)$ is linear and we have two points, $x_\ell(t_1)$ and $x_\ell(t_2)$ such that $t_1 < t^* < t_2$, then if the directions have been normalized

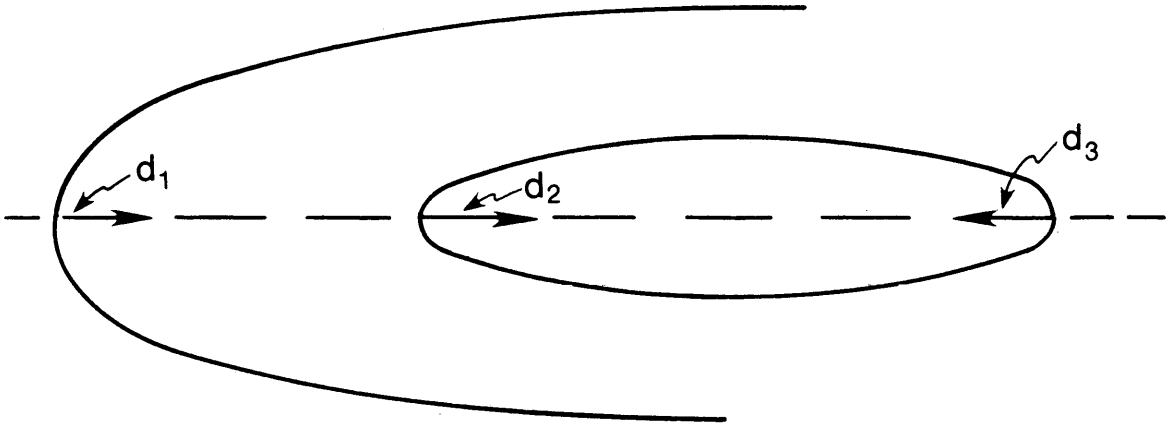

Figure 3. Determining if the minimum of a Hypervalley is Bracketed.

ARTHUR LAKES LIBRARY
COLORADO SCHOOL of MINES
GOLDEN, COLORADO 80401

$d_{V_\ell}(x_\ell(t_1)) \cdot d_{V_\ell}(x_\ell(t_2)) = -1.0$ and we say that the minimum of the hypervalley $x_\ell(t)$ is bracketed by $x_\ell(t_1)$ and $x_\ell(t_2)$. In practice, hypervalleys are not linear. Therefore, we use

$$d_{V_\ell}(x_\ell(t_1)) \cdot d_{V_\ell}(x_\ell(t_2)) < \beta \tag{14}$$

where $-1.0 < \beta < 0.0$, as our criterion to determine if the minimum on a hypervalley is bracketed. This becomes our loose convergence criterion for a hypervalley where we have found that $\beta = -0.7$ works well. Other values of β or another convergence criterion may improve the performance of the algorithm. For simplicity of notation in the description of the algorithm, we denote the previous $d_{V_\ell}(x_\ell(t))$ by d_{LAST}.

THE ALGORITHM

x = starting point

$\ell = 1$

Compute the gradient, Hessian, eigenvectors and eigenvalues at x

$d_{LAST} = 0$

$C_l(x) = \{i \varepsilon I, \lambda_i \geq \gamma \lambda_l\}$

$V_l(x) = \{i \varepsilon C_l^c, \lambda_i \geq \gamma \lambda_{m_l+1}\}$

WHILE NOT converged on subvariety $S_I(x)$ DO (test overall convergence)

BEGIN

WHILE NOT converged on $S_{C_\ell}(x)$ DO

BEGIN (minimize on hypercross section to find hypervalley)

Minimize $f(x + \alpha d_{C_\ell}(x))$ to get a new x

Update gradient, Hessian, eigenvectors and eigenvalues at x

END

IF $d_{V_\ell}(x) \cdot d_{LAST} < \beta$ THEN (see Figure 3)

BEGIN (hypervalley is bracketed so start new stage)

$C_{\ell+1} = C_\ell \cup V_\ell$

x = point on $x_\ell(t)$ with smallest function value

Get $V_{\ell+1}$

Update gradient, Hessian, eigenvectors and eigenvalues at x

$\ell = \ell + 1$

$d_{LAST} = 0$

END

ELSE

BEGIN (step along the hypervalley until an increase in f(x))

$d_{LAST} = d_{V_\ell}(x)$

Step along $d_{V_\ell}(x)$ until f(x) increases to get a new x

Update gradient, Hessian, eigenvectors and eigenvalues at x

END

END OF THE ALGORITHM

If the hypervalley is linear, the direction $d_{V_\ell}(x)$ is searched to find the minimum along the hypervalley and then the algorithm sets up a new stage. The check for linearity is done at the new x obtained after stepping along $d_{V_\ell}(x)$ until an increase in the objective function is obtained.

MODIFICATION OF THE ACCELERATION ALGORITHM

The set V_ℓ can often be included with the set C_ℓ when minimizing on the hypercross section to obtain a modified direction which gives faster acceleration along the hypervalley. This is clearly illustrated in two dimensions in Figure 4 where direction $d_{C_\ell}(x)$ is shown in direct contrast with the modified direction $d_{C_\ell \ U \ V_\ell}(x)$. But, $d_{V_\ell}(x)$ is not included if its dot product with the previous $d_{V_\ell}(x)$ direction, d_{LAST}, is negative because the minimum on the hypervalley may be bracketed. If Newton's method predicts a large change along d_{V_ℓ}, then V_ℓ can be included in C_ℓ, but its influence is restricted so that it does not dominate the eigenvectors already in $d_{C_\ell}(x)$. For the current algorithm, if

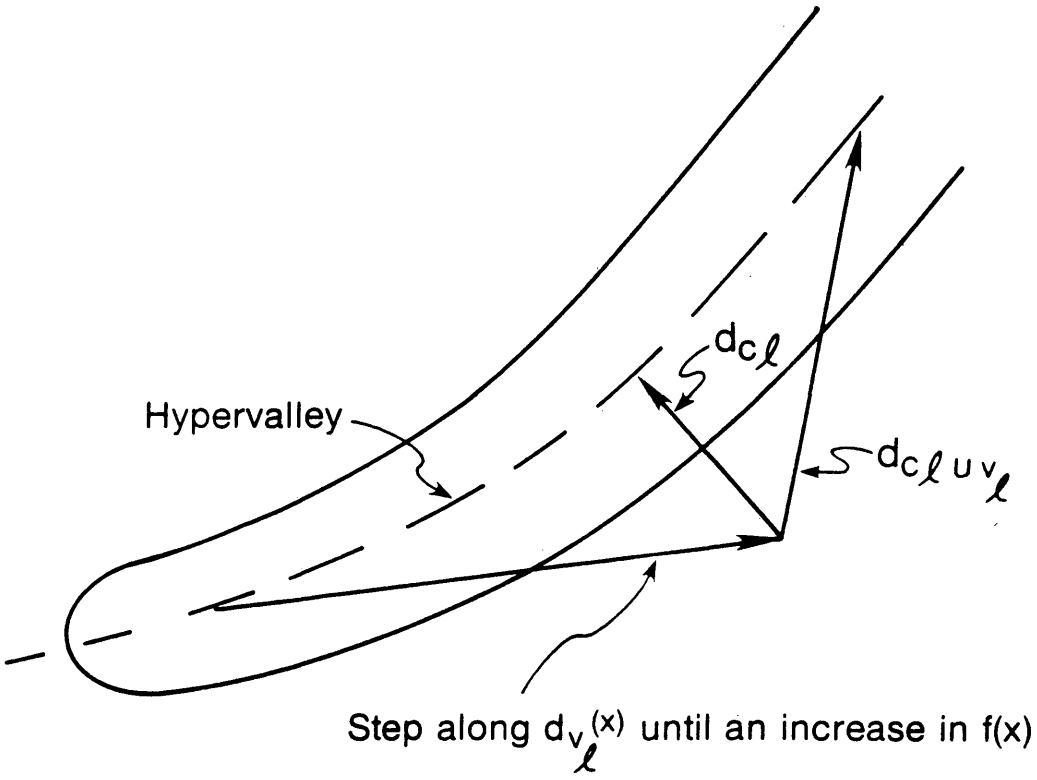

Figure 4. Illustration of the Modified Accleration Procedure.

$$||d_{V_\ell}(x)|| > 100\ ||d_{C_\ell}(x)||\quad \text{then}$$

$$d_{V_\ell}(x) = d_{V_\ell}(x)\,\frac{||d_{C_\ell}(x)||}{||d_{V_\ell}(x)||} \tag{15}$$

before being added to $d_{C_\ell}(x)$.

OBTAINING EIGENVECTORS AND EIGENVALUES

We used two different methods of obtaining eigenvectors and eigenvalues. The first (Appendix B) used Householder's method [10,11] as implemented in the IMSL subroutine EIGRS. The second method used power iterations [12] to obtain eigenvectors and eigenvalues. This was done in an attempt to use previous information to reduce the computer time needed to calculate the eigenvectors and eigenvalues.It was assumed that, most of the time, the eigenvectors would not change direction appreciably from the last time they were evaluated. Therefore, the previous eigenvectors would serve as good initial guesses for the next set of power iterations thus requiring only a few iterations for convergence. And,

the current gradient gives a good initial guess for the eigenvectors when they are first estimated. The initial guess should be perpendicular to the previous eigenvectors already obtained which the gradient is because the objective function has been minimized over the subvariety containing these eigenvectors. Also, we want eigenvectors with the largest eigenvalues which, as pointed out previously, point essentially in the negative gradient direction. Finally, the power method obtains eigenvectors with the largest eigenvalues first. Therefore, computer time is not wasted computing eigenvectors that are not needed.

This method worked very well in some cases and not at all in others. The problem was in the deflation technique used. Once one eigenvector is obtained, it is eliminated from the Hessian using the following deflation method [12]

$$H_1 = H - \lambda_1 e_1 e^T_1 \tag{16}$$

H_1 is then used to obtain the second eigenvector. This process is repeated until all the desired eigenvectors are obtained. The problem arises with the last eigenvectors

calculated. If the first eigenvectors and eigenvalues are not calculated accurately enough, the deflation procedure will give an inaccurate deflated matrix due to round off errors. This results in the last eigenvectors calculated pointing in the wrong direction. And, as it turns out, these eigenvectors are the most critical to the proper working of the algorithm because these eigenvectors are used to determine the hypervalley. This problem is compounded when the ratio of the largest and smallest eigenvalues becomes large. In this case, even a 48 bit machine and a large number of power iterations are not sufficient to solve the problem.

STANDARD TEST PROBLEMS

BANANA was tested using several standard test problems. These test problems are given in Table 1 along with the reference where they were obtained. We also designed several test problems of our own in order to obtain some difficult tests with a higher number of variables. Test problems 1, 2, and 3 are extensions of Rosenbrock's function, number 4. When N = 2 and MAX = 100 in test problem number 3, Rosenbrock's test problem is obtained.

Table 1. Standard Objective Functions

No.	FUNCTION	Starting Point	Reference
1	$(1-x_1)^2 + \sum_{i=2}^{N} [\frac{(i-1)(MAX-1)}{N-1}] (X_i - X_{i-1}^2)^2$, N=12, MAX = 100	(-1.5,0.8,...,0.8)	*
2	SAME AS NO. 1 WITH, N=6, MAX = 100	(-1.5,0.8,...,0.8)	*
3	$225\ (x_4-x_3^2)^2 + 100\ (x_3-x_2^2)^2 + 25\ (x_2-x_1^2)^2 + (1-x_1)^2$	(-1.5, 1, 1, 1)	*
4	$100\ (x_2-x_1^2)^2 + (1-x_1)^2$	(-1.2, 1.0)	[13]
5	SAME AS NO. 4	(-2.547, 1.489)	[13]
6	$225\ (x_3-2x_2^2)^2 + 100\ (x_2-(x_1-1/2)^2 + 1/4)^2 + x_1-1.5)^2$	(-1.5, 0.707, 1)	*
7	$(x_1 + 10\ x_2)^2 + 5(x_3-x_4)^2 + (x_2-2x_3)^4 + 10(x_1-x_4)^4$	(-3, -1, 0, 1)	[14]
8	SAME AS NO. 7	(1, 1, 1, 1)	[14]
9	$100\ (x_2-x_1^3)^2 + (1-x_1)^2$	(-1.2, 1.0)	[15]
10	SAME AS NO. 9	(0.248, -3.082)	[15]
11	$(x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$	(1, 1)	[1]
12	$100(x_3 - (\frac{x_1 + x_2}{2})^2)^2 + (1-x_1)^2 + (1-x_2)^2$	(-1.2, 2.0)	[1]
13	$(1.5-x_1(1-x_2))^2 + (2.25 - x_1\ (1-x_2^2))^2 + (2.625-x_1(1-x_2^3))^2$	(8.0, 0.8)	[16]
14	SAME AS NO. 13	(0, 0)	[16]
15	$(x_2-x_1^2)^2 + (1-x_1)^2$	(-1.2, 1.0)	[15]
16	SAME AS NO. 15	(0.211, 3.505)	[15]
17	$(x_2-x_1^2)^2 + 100\ (1-x_1)^2$	(-1.2, 1.0)	[15]
18	$4\ (x_1-5)^2 + (x_2-6)^2$	(8, 9)	[1]
19	$225\ x_3^2 + 100\ x_2^2 + x_1^2$	(-5, -3, 1)	*

*Test functions designed by the author.

These test problems were also run using the IMSL conjugate gradient [2] subroutine ZXCGR and the IMSL quasi-Newton [3] subroutine ZXMIN. A modified Newton method [1], where Newton's direction is searched as described in Appendix A, was also included to give an indication of the difficulty of the problem. The computer these tests were run on was a Burroughs B7900 using single precision with a 48 bit representation. To obtain a fair comparison, all the algorithms were terminated after the first function call with a function value less than or equal to 10^{-13}. Analytical derivatives were used for all algorithms except quasi-Newton which uses forward differences to estimate derivatives until it gets close to the minimum and then switches to central differences. To be consistent, we calculated the number of function evaluations assuming that a two point forward difference was used to obtain the gradient. This was done because this is how the gradient is obtained in our oil well problem. This adjustment was not done for quasi-Newton as it takes its own numerical derivatives.

Table 2 gives a summary of the number of function evaluations required by each algorithm to solve the test problems. Appendix C gives more detailed results of these tests.

The extra work BANANA does to accelerate along the hyper-valleys is not needed in solving a purely quadratic problem, e.g., functions 18 and 19 in Table 1, where the modified Newton algorithm does very well. For problems of medium difficulty, all the algorithms perform about the same. The modified Newton method predicted an up hill direction at some point during minimization for functions 1, 11, 13 and 14 and thus was not able to find the minimum. Quasi-Newton quit on function 1 after 32077 function evaluations with a function value of $4.0*10^{-11}$ because of rounding errors.

As expected, BANANA did not do as well in the comparison of computer times. For the simple test cases, BANANA took two to five times longer than the other algorithms. The conjugate gradient algorithm took the least amount of computer time in almost every test problem. The reason for this is that it is a simple algorithm requiring little work between function evaluations.

Table 2. Number of function evaluations needed to solve the test functions.

test function	BANANA	modified Newton	conjugate gradient	quasi-Newton
1	2177	*	59801	**
2	407	667	2472	1239
3	222	354	1201	594
4	72	92	223	209
5	33	127	220	116
6	169	22400	824	382
7	235	139	566	265
8	228	141	1551	213
9	51	133	196	110
10	106	88	196	139
11	55	*	43	69
12	50	49	149	116
13	40	*	160	111
14	46	*	70	79
15	39	32	67	68
16	39	34	55	56
17	19	23	37	52
18	17	8	13	47
19	30	9	31	71

* Algorithm quit because Newton's direction did not give a decrease in the objective function.
** Algorithm quit due to rounding errors.

THE OIL WELL PROBLEM

The specific problem behind the development of this algorithm is the estimation of oil reservoir properties [17-39] from well test data. An oil reservoir consists of a layer of porous rock deep under ground with oil under pressure in the pores. Usually, many such layers are found. The oil is produced by drilling a well into the rock. The oil flows radially through the rock to the well, then up the well to the surface, Figure 5.

The equation used to simulate the flow of oil in the reservoir is:

$$\frac{1}{r}\frac{\partial}{\partial r}\left[rT\left(\frac{\partial p}{\partial r}\right)\right] + \frac{\partial}{\partial Z}\left[T\left(\frac{\partial p}{\partial Z}\right)\right] = \frac{\partial}{\partial t}\left(\phi\rho\right) \qquad (17)$$

where

T = mobility = $0.0063228\ k/\mu$

p = reservoir pressure (psi)

ϕ = reservoir porosity (fraction), a function of p

ρ = oil density (lbm/cu ft) a function of p

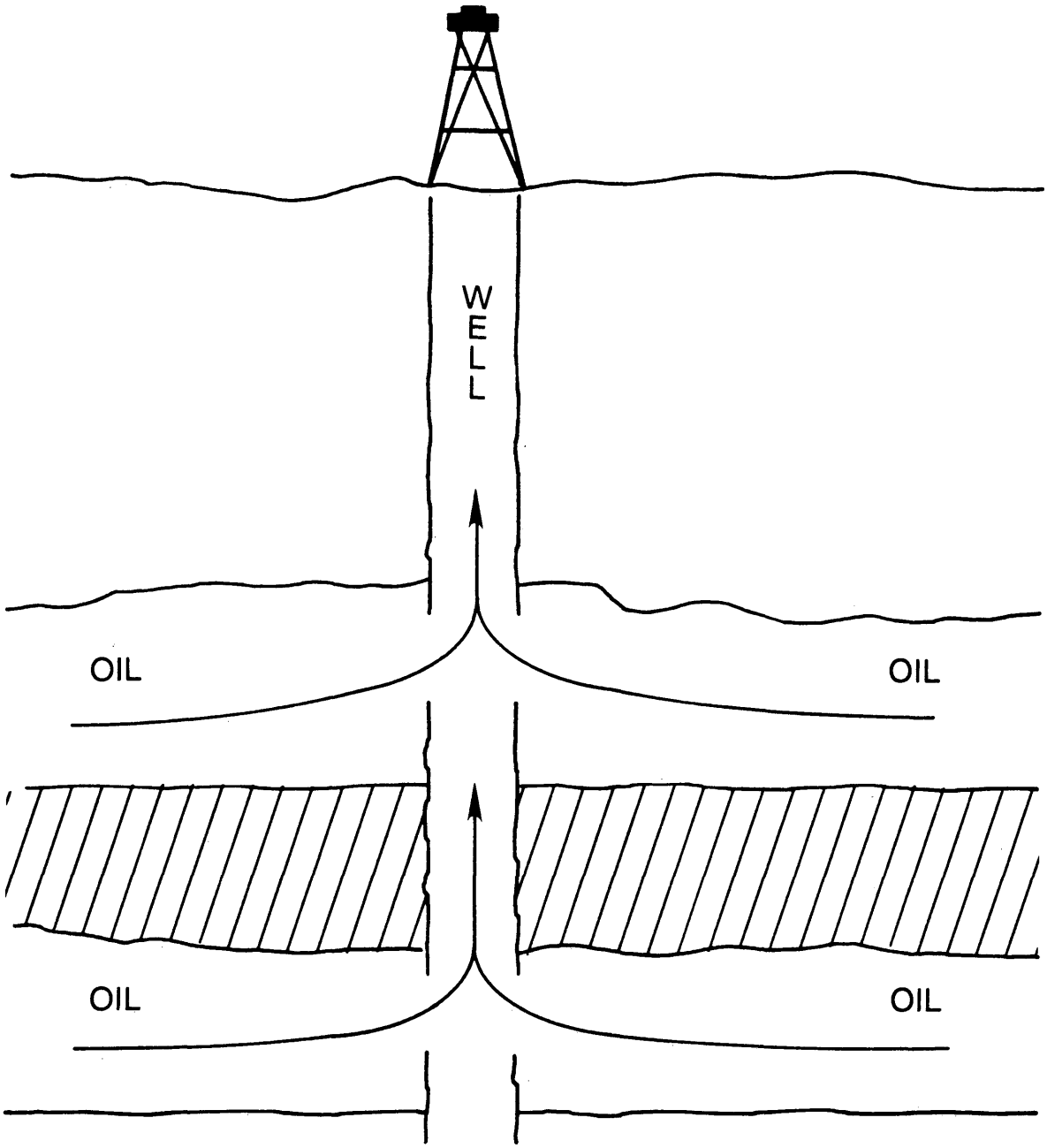

Figure 5. Example Oil Reservoir.

μ = oil viscosity (cp) a constant

k = reservoir permeability (md)

t = time (days)

Equation 17 is derived using Darcy's law for flow in porous media. Tables of porosity and density as a function of pressure are provided as input. Equation 17 is discretized to obtain the finite difference equation

$$\frac{r_{i+1/2,j}}{r_{i,j}} \frac{T^n_{i+1/2,j}}{\Delta r_{i,j}} \left[\frac{p^{n+1}_{i+1,j} - p^{n+1}_{i,j}}{\Delta r_{i+1/2,j}} \right] -$$

$$\frac{r_{i-1/2,j}}{r_{i,j}} \frac{T^n_{i-1/2,j}}{\Delta r_i} \left[\frac{p^{n+1}_{i,j} - p^{n+1}_{i-1,j}}{\Delta r_{i-1/2,j}} \right] +$$

$$\frac{T^n_{i,j+1/2}}{\Delta Z_{i,j}} \left[\frac{p^{n+1}_{i,j+1} - p^{n+1}_{i,j}}{\Delta Z_{i,j+1/2}} \right] -$$

$$\frac{T^n_{i,j-1/2}}{\Delta Z_{i,j}} \left[\frac{p^{n+1}_{i,j} - p^{n+1}_{i,j-1}}{\Delta Z_{i,j-1/2}} \right]$$

$$= \frac{\phi^{n+1}_{i,j}\ \rho^{n+1}_{i,j} - \phi^{n}_{i,j}\ \rho^{n}_{i,j}}{\Delta t} \qquad (18)$$

where ΔZ and Δr are grid block sizes in the vertical and radial direction and Δt is the time step size. The subscript indicates the grid block, i for the radial direction, Figure 6, j for the vertical direction. The superscript indicates the time level, n for the current time level, n+1 for the next time level. The index $i \pm 1/2$ and $j \pm 1/2$ on the mobility indicates that it is evaluated at the pressure of the upstream node.

The production or injection rate, Q (Bbl/day), is specified at the well as a function of time. Therefore, the boundary condition becomes

$$\hat{q}_j = \frac{r_{1/2,j}}{r_{1,j}} \frac{T_{1/2,j}}{\Delta r_{1,j}} \left[\frac{P_{1,j} - \hat{P}_w}{\Delta r_{1/2,j}} \right], \quad j = 1, \ldots, \text{JMAX} \qquad (19)$$

$$Q = \sum_j \hat{q}_j \qquad (20)$$

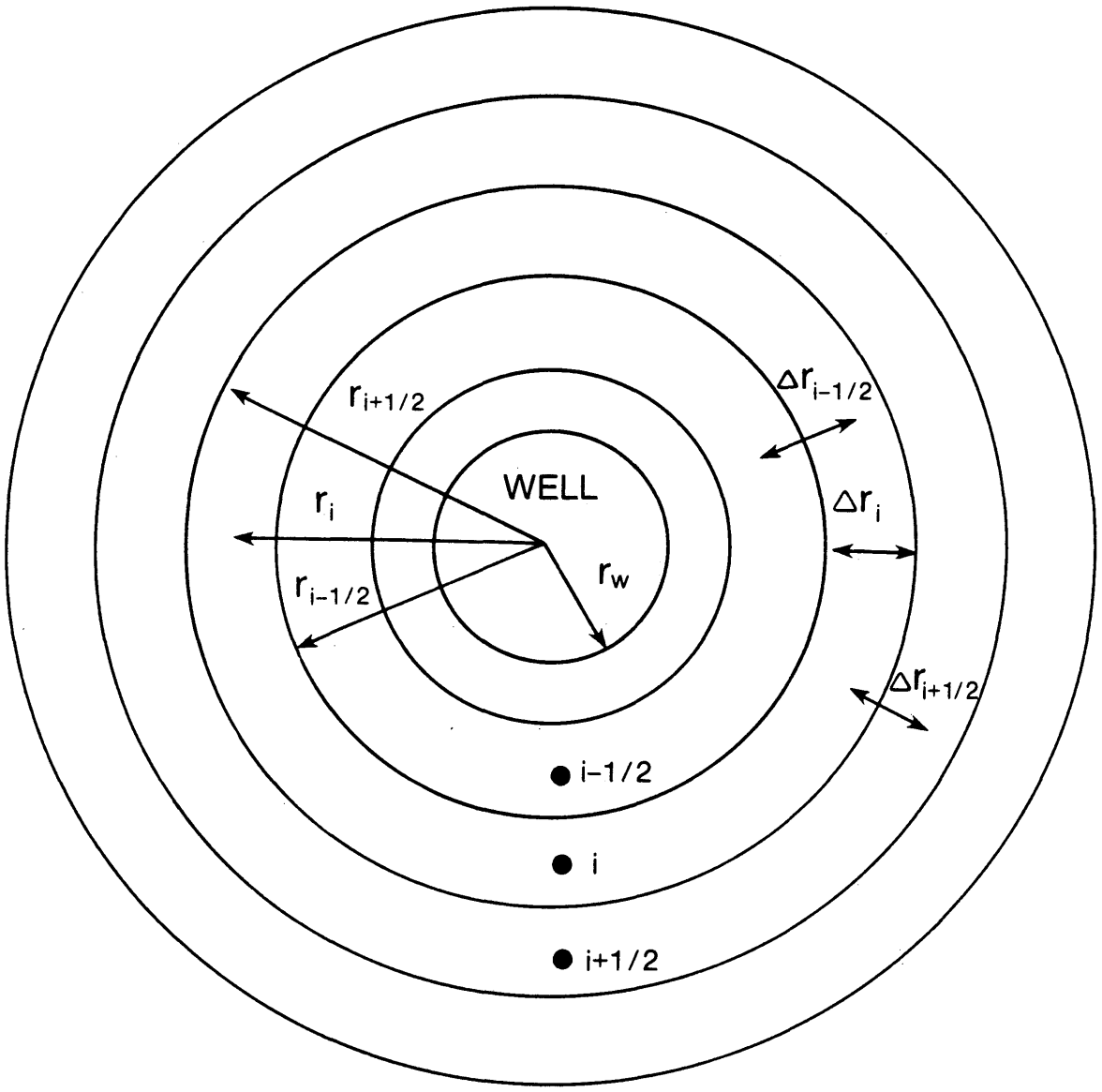

Figure 6. Areal View of the Cylindrical Coordinate System.

where $\hat{p}_w$ is the well pressure. There is a no flow boundary condition at the outside edge of the reservoir, therefore we have

$$T_{IMAX+1/2,j} = 0 \qquad j = 1, \ldots, JMAX \tag{21}$$

$$T_{i,1/2} = 0 \quad , \quad i = 1, \ldots, IMAX \tag{22}$$

$$T_{i,JMAX+1/2} = 0 \qquad i = 1, \ldots, IMAX \tag{23}$$

Skin factor, s, is related to permeability at a radius of r_s by the equation

$$s = (k/k_s - 1)\ln(r_s/r_w) \tag{24}$$

where r_s is a constant specified on input.

Given values for k, ΔZ, p_{init} and s, the nonlinear system of equations (18-24) are solved to obtain $\hat{p}_w$ and $\hat{q}_j$ as a function of time. The residual sum of squares objective function then becomes

$$f(k,\Delta Z,p_{init},s) = \sum_{\xi}(p_{w,\xi} - \hat{p}_w(k,\Delta Z,p_{init},s,t_\xi))^2 +$$

$$\sum_{\xi}\sum_{j}(q_{j,\xi} - \hat{q}_j(k,\Delta Z,p_{init},s,t_\xi))^2 \qquad (25)$$

where $p_{w,\xi}$ and $q_{j,\xi}$ are the observed data values.

BANANA, conjugate gradient [2], quasi-Newton [3] and modified Newton [1] were run on four well test data problems. The IMSL Marquardt subroutine [5-8], ZXSSQ, was also run using Brown's algorithm without strict descent (IOPT = 0) and with strict descent using the IMSL values for the input parameters (IOPT = 1). Data for the first three well test problems were obtained by using known values for k, ΔZ, p_{init} and s. Equations (18-24) were solved to get $\hat{p}_w$ and $\hat{q}_j$ as a function of time. These values of $\hat{p}_w$ and $\hat{q}_j$ were then used as the observed data, $p_{w,\xi}$ and $p_{j,\xi}$ in (25) to see if we obtain our original values for k, ΔZ, p_{init} and s at the minimum of the objective function. The first two test problems were taken directly from this data with two different starting points. Fifteen variables were adjusted, five permeabilities, five skin factors,

and five initial pressures. The third test problem is the same as the second except that random errors were added to the simulated data. The fourth test problem was taken from real data where twelve variables were used. More information on these oil well test problems is given in Appendix E.

The number of function evaluations each algorithm took to solve these test problems is given in Table 3. For the real data test problem, number 4, modified Newton was at 187 function evaluations with a function value of $7.11*10^5$ when it went uphill. The quasi-Newton algorithm was at 87 function evaluations and a function value of $4.1*10^6$ shortly before the objective function became too large causing the computer to terminate the run. Both Marquardt methods terminated due to too large of an objective function value after only a few function evaluations. BANANA had a function value of $6.1*10^5$ after 50 function evaluations. The initial function value is $1.0*10^7$. The minimum function value obtained was $5.2*10^5$. Each of these runs took an average of 10 hours on the B7900. This is equivalent to about 2 hours on an IBM 3081 or about 1/2 hour on a Cray. Therefore, the optimization algorithm used must be as efficient as possible.

Table 3. Number of function evaluations needed to solve the well test problems.

	problem			
	1	2	3	4
BANANA	380	780	425	269
Modified Newton	69	*	***	*
Conjugate Gradient	***	***	***	***
Quasi-Newton	626	646	***	**
Marquardt (IOPT = 0)	81	**	129	**
Marquardt (IOPT = 1)	**	***	***	**

* Algorithm was unable to solve the problem because it picked an up hill direction.

** Algorithm was terminated due to an objective function value that was too large.

*** Algorithm used 10 hours on the Burroughs B7900 computer but did not reduce the objective function by an appreciable amount.

The addition of random errors does not appear to increase the difficulty of the problem as much as first thought. In fact, just changing the starting point appears to have just as much effect. Modified Newton and Marquardt with Brown's algorithm do very well on problem 1, indicating that it should be an easy problem. However, after moving the starting point, number 2, both these algorithms fail to solve the problem. And, it is significant to note that BANANA is the only algorithm that successfully solved all four test problems and was also the only algorithm to successfully solve the real data test problem, number 4.

CONCLUDING COMMENTS

The BANANA algorithm successfully obtained the minimum of all the objective functions we tested. Therefore, it appears to have good global characteristics. It may be possible to reduce the number of function evaluations required by using a different criterion for convergence on a hypervalley or hypercross section. The grouping criterion for the eigenvetors is also very critical to the operation of the algorithm. BANANA does very well at the start of the

minimization process but slows down when approaching the minimum. Since Newton's method does best near the minimum, faster convergence may be obtained by switching to Newton's method earlier. Therefore, an eigenvector grouping criterion that includes more eigenvectors in the groups with smaller eigenvalues may help improve convergence.

Using a more cost effective method of obtaining eigenvectors and eigenvalues will make BANANA useful for a larger number of practical problems. Switching to Newton's method earlier may enable the power method of obtaining eigenvectors and eigenvalues to be used. The power method works well at the start of the minimization process when the eigenvalues are illconditioned but breaks down near the end of the minimization when the last few eigenvectors are required. Therefore, if BANANA switches to Newton's method before these last few eigenvectors are required, then the power method may be usable.

NOMENCLATURE

C	Hypercross section index set.
DS	Step size.
d	Search or step direction vector.
DS13	Distance from the first to third point in a three point pattern obtained during a line search.
E(x)	Matrix of eigenvectors, $E = \{e_1, \ldots, e_n\}$.
EPS	Tolerance used for the line search.
e	Eigenvector.
f(x)	Objective function.
g(x)	Gradient of the objective function.
H(x)	Hessian of the objective function.
I	Index set, $I = \{1, \ldots, n\}$.
IMAX	Number of radial nodes in the model.
i	Variable index.
JMAX	Number of layers in the model.
k	Newton algorithm stage index.
k	Reservoir permeability.
ℓ	Stage index.
m	Number of elements in C.
n	Number of optimization variables.
o	Number of elements in J.

P	A general subset of index set I.
p	Reservoir pressure, psi.
Q	Well flow rate, Bbl/day.
q	Well layer flow rate, Bbl/day.
R^n	Space of real n-tuples with Euclidian norm.
r	Radial distance measured from the well, ft.
$S_P(x)$	Subvariety of R^n defined by the index set P as in Definition 1.
S	A subspace.
s	Skin factor.
T	Mobility.
t	Time, days.
t	Parameter used in the differential equation that defines the hypervalley.
x	A point in R^n.
Z	Vertical distance, ft.
α	Linear search parameter.
β	Constant used to determine if the minimum on a hypervalley has been bracketed, $-1 < \beta < 0$, $\beta = -0.7$.
γ	A constant used to determine the eigenvector directions to be included in S_{V_ℓ}, $0 < \gamma < 1$, $\gamma = 1/2$.

$\lambda_i(x)$	Eigenvalue of the Hessian of the objective function.
$\Lambda(x)$	Diagonal matrix of eigenvalues.
μ	Oil viscosity, cp.
ϕ	Reservoir porosity, fraction.
ρ	Oil density, lbm/cu ft.
τ	Convergence tolerance, $0 < \tau$, $\tau = 0.01$.

SUPER SCRIPTS

c	Complement of the set.
^	Calculated data.
~	Indicates variable is expressed in the coordinates for the eigenvector basis of R^n.
*	Indicates that the variable is at the minimum of f(x) on a line or subvariety.

SUBSCRIPTS

i	Radial grid block index.
init	At zero time.

LAST	At previous point on the hypervalley.
w	At the well.
s	At a grid block whose permeability is modified to simulate skin.
ξ	Observed data index.
1	Hessian has been deflated using the first eigenvector.

REFERENCES

1. Himmelblau, D. M.: Applied Nonlinear Programming, McGraw-Hill, New York (1972).

2. Powell, M. J. D.: "Restart Procedures for the Conjugate Gradient Method," Mathematical Programming, Vol. 12, pp. 241-254 (1977).

3. Fletcher, R.: "Fortran Subroutines for Minimization by Quazi-Newton Methods," Report R7125 AERE, Harwell, England (June, 1972).

4. Jones, A.: "SPIRAL - A New Algorithm for Non-Linear Parameter Estimation Using Least Squares," Computer J., Vol. 13, No. 3, pp. 301-308 (August 1970).

5. Brown, K. M., and Dennis, J. E.: "Derivative Free Analogues of the Levenberg-Marquardt and Gauss Algorithms for Nonlinear Least Squares Approximations," Numerische Mathematik, Vol. 18, pp. 289-297 (1972).

6. Brown, K. M.: "Computer Oriented Methods for Fitting Tabular Data in the Linear and Nonlinear Least Squares Sense," Department of Computer, Information and Control Sciences, TR No. 72-13, University of Minnesota (1972).

7. Levenberg, K.: "A Method for the Solution of Certain Non-Linear Problems in Least Squares," Quart. Appl. Math., 2, pp. 164-168 (1944).

8. Marquardt, D. W., "An Algorithm for Least-Squares Estimation of Nonlinear Parameters," J. SIAM, Vol. 11, No. 2 (1963).

9. Luenberger, D. B.: Linear and Nonlinear Programming, Addison-Wesley, Reading, Massachusetts (1984).

10. Wilkinson, J. H.: The Algebraic Eigenvalue Problem, Clarendon Press, Oxford (1965).

11. Smith B. T., Boyle, J. M., Garbow, B. S., Ikebe, Y., Klema, V. C., and Moler, C. B.: Matrix Eigensystem Routines, Springer-Verlag (1974).

12. Froberg, C. E.: Introduction to Numerical Analysis, Addison-Wesley, Reading, Massachusetts (1965).

13. Rosenbrock, H. H.: "An Automatic Method for Finding the Greatest and Least Value of a Function," Computer J., Vol. 3, pg. 175 (1960).

14. Powell, M. J. D.: "An Iterative Method for Finding Stationary Values of a Function of Several Variables," Computer J., Vol. 5, pg. 147 (1962).

15. Whitte, B. F. W. and Holst, W. R.: "Two New Direct Minimum Search Procedure for Functions of Several Variables," 1964 Spring Joint Computer Converence, Washington D. C. (1964).

16. Beale, E. M. L.: "On an Iterative Method of Finding a Local Minimum of a Function of More Than One Variable," Princeton Univ. Stat. Tech. Res. Group Tech. Rept. 25 (November, 1958).

17. Jacquard, P. and Jain, C.: "Permeability Distribution from Field Pressure Data," Society of Petroleum Engineers Journal, pp. 281-294 (December 1965).

18. Jahns, H. O.: "A Rapid Method for Obtaining a Two-Dimensional Reservoir Description from Well Pressure Response Data," Society of Petroleum Engineers Journal, pp. 325-327 (December 1966).

19. Coats, K. H., Dempsey, J. R., and Henderson, J. H.: "A New Technique for Determining Reservoir Description from Field Performance Data," Society of Petroleum Engineers Journal, pp. 66-74 (March 1970).

20. Slater, G. E. and Durier, E. J.: "Adjustment of Reservoir Simulation Models to Match Field Performance," Society of Petroleum Engineers Journal pp. 295-305 (September 1971).

21. Thomas, K. L., Hellums, L. J., and Reheis, G. M.: "A Nonlinear Automatic History Matching Technique for Reservoir Simulation Models," Society of Petroleum Engineers Journal, pp. 508-514 (December 1972).

22. Veatch, R. W. Jr. and Thomas, G. W.: "A Direct Approach for History Matching," paper SPE 3515 presented at the SPE 46th Annual Fall Meeting, New Orleans (October 3-6, 1971).

23. Carter, R. D., Kemp, L. F., and Williams, D.: "Performance Matching with Constraints," Society of Petroleum Engineers Journal, pp. 187-196 (April 1974).

24. Chen, W. H., Gavalas, G. R., Seinfeld, J. H., and Wasserman, M. L.: "A New Algorithm for Automatic History Matching," Society of Petroleum Engineers Journal, pp. 593-608 (December 1974).

25. Soloranzo, L. N. and Arrendondo, S. E., "Method for Automatic History Matching of Reservoir Simulation Models," paper SPE 4594 presented at the SPE 48th Annual Fall Meeting, Las Vegas (September 30 - October 2, 1973).

26. Boberg, T. C., Woods, E. G., and McDonald, W. J., Jr.: "Application of Inverse simulation to a Complex Multi-Reservoir System," Journal of Petroleum Technology, pp. 80-808 (July 1974).

27. Chavent, C., Dupuy, M., and Lemonnier, P.: "History Matching by Use of Optimal Control Theory," Society of Petroleum Engineers Journal, pp. 74-86 (February 1975).

28. Chen, W. H. and Seinfeld, J. H.: "Estimation of the Location of the Boundary of a Petroleum Reservoir," Society of Petroleum Engineers Journal, pp. 19-38 (February 1975).

29. Wasserman, M. L., Emanuel, A. S., and Seinfeld, J. H.: "Practical Applications of Optimal Control Theory to History Matching Multiphase Simulator Models," Society of Petroleum Engineers Journal, pp. 347-355 (August 1975).

30. Dougherty, E. L. and Khairklah, D.: "History Matching of Gas Simulation Models Using Optimal Control Theory," paper SPE 5371 presented at the SPE 45th Annual California Regional Meeting, Ventura (April 2-4, 1975).

31. Wasserman, M. L. and Emanuel, A. S.: "History Matching Three Dimensional Models Using Optimal Control Theory," paper number 7611 presented at the 27th Annual Technical Meeting of the Petroleum Society of CIM, Calgary (1976).

32. Gavalas, G. R., Shah, P. C., and Seinfeld, J. H.: "Reservoir History Matching by Bayesian Estimation," Society of Petroleum Engineers Journal, pp. 337-350 (December 1976).

33. Van den Bosch, B. and Seinfeld, J. H.: "History Matching in Two-Phase Petroleum Reservoirs: Incompressible Flow," Society of Petroleum Engineers Journal, pp. 398-406 (December 1977).

34. Shah, P. C., Gavalas, G. R. and Seinfeld, J. H.: "Error Analysis in History Matching: The Optimum Level of Parameterization," Society of Petroleum Engineers Journal, pp. 219-228 (June 1978).

35. Chavent, G. Cohen, G. and Espy, M.: "Determination of Relative Permeabilities and Capillary Pressures by an Automatic Adjustment Method," paper SPE 9237 presented at the 55th Annual Fall Technical Conference and Exhibition, Dallas (September 21-24, 1980).

36. Dogru, A. H. and Seinfeld, J. H.: "Comparison of Sensitivity Coefficient Calculation Methods in Automatic History Matching," paper SPE 8251 presented at the 54th Annual Fall Technical Conference and Exhibition, Las Vegas (September 23-26, 1979).

37. Dogru, A. H. and Seinfeld, J. H.: "Design of Well Tests to Determine the Properties of Stratified Reservoirs," paper SPE 7694 presented at the Fifth Symposium on Reservoir Simulation, Denver (February 1-2, 1979).

38. Welty, D. H. and Miller, W. C.: "Automated History Matching of Well Tests," SPE 7695 presented at the Fifth SPE Symposium on Reservoir Simulation, Denver, Colorado (January 31-Fegruary 2, 1979).

39. Watson, A. T., Gavalas, G. R., and Seinfeld, J. H.: "Identification of Estimates of Two-Phase Reservoir Properties in History Matching," Society of Petroleum Engineers Journal, pp. 697-706 (December 1984).

APPENDIX A

Step Size for Linear Searches in Minimization of a Hypercross Section

The proper choice of an initial step size for linear searches along Newton's direction is very important in the proper implementation of an algorithm. It is especially important in this case as too large a step size will result in the computer terminating the program prematurely. Therefore, Newton's step size, DS, is modified using the following equation:

$$DS = \sqrt{0.1 * DS} \; . \qquad \text{(A-1)}$$

If DS < 0.1, then it is used as is without any modification. Once a three point pattern has been obtained, quadratic interpolation is used to refine the estimate of the minimum. This process is terminated when the new predicted minimum is less than a distance of EPS away from the last one. EPS is defined as:

$$EPS = MIN(DS13 \; / \; 100 \; , \; 0.005) \qquad \text{(A-2)}$$

where DS13 is the distance from the first to the third point of the three point pattern when it is first established. This gives a tighter tolerance on the line search as the minimum is approached. And, the loose tolerance when the algorithm is away from the minimum prevents excessive function evaluations from being performed. When very close to the minimum, a pure Newton method is faster than searching Newton's direction. Therefore, if DS < 0.01 and this DS results in a point giving a reduced objective function, then no line search is performed this point being taken as the minimum of the line search.

APPENDIX B

Listing of FORTRAN Version of BANANA

```
      COMMON /IUNIT/ NIN, NOUT
      COMMON /STAT/ CPUT,NFUNC,NGRAD,NFLINE,NLINE,NFL,NWNTN
      DIMENSION DIROLD(4),DIR(4),X(4),XW(4),X0(4),GRAD(4),
      DIMENSION EVAL(4),U(4),EWORK(4),E(16),DELS(4),H(16)
      NIN = 5
      NOUT = 6
      NFUNC = 0
      NGRAD = 0
      NFLINE = 0
      NLINE = 0
      NWNTN = 0
      M = 4
      X0(1) = -1.5
      X0(2) = 1.0
      X0(3) = 1.0
      X0(4) = 1.0
      CPUT = TIME(12)
      CALL BANANA(DIROLD,DIR,X,XW,M,F0,X0,GRAD,H,EVAL,U,
     $    EWORK,E,DELS)
      END
      LOGICAL FUNCTION ACCPOS(DIROLD,NEW,KV,K,NORM,DIR,X,XW,
     $    DSBASE,DS12,DS23,DS,NEWXW,M,LINEAR,N)
C
C    PERFORM CALCULATIONS NEEDED TO SET UP THE ACCELERATION
C    ALONG THE VALLEY.
C
      DIMENSION DIROLD(M),DIR(M),X(M),XW(M)
      COMMON /STAT/ CPUT,NFUNC,NGRAD,NFLINE,NLINE,NFL,NWNTN
      REAL NORM
      LOGICAL NEW,NEWXW,LINEAR
      DO 10 I = 1,M
   10 DIROLD(I) = 0.0
      ACCPOS = .FALSE.
      NEWXW = .TRUE.
      LINEAR = .FALSE.
      N = 1
      IF (NEW .OR. KV .LE. K .OR. NORM .EQ. 0.0) GO TO 30
C
C    SET UP FOR ACCELERATION.
C
      DO 20 I = 1,M
      DIROLD(I) = DIR(I)
   20 X(I) = XW(I)
      DSBASE = 0.0
```

ARTHUR LAKES LIBRARY
COLORADO SCHOOL of MINES
GOLDEN, COLORADO 80401

```
      DS12 = 0.0
      DS23 = 0.0
      DS = NORM
      IF (NORM .GT. 0.1) DS = SQRT(0.1 * NORM)
      NFL = 0
      ACCPOS = .TRUE.
   30 CONTINUE
      RETURN
      END
      SUBROUTINE BANANA(DIROLD,DIR,X,XW,M,F0,X0,GRAD,H,
     $    EVAL,U,EWORK,E,DELS)
      DIMENSION DIROLD(M),DIR(M),X(M),XW(M),X0(M),GRAD(M)
      DIMENSION EWORK(M),DELS(M),EVAL(M),U(M),H(M,M)
      DIMENSION E(M,M)
      COMMON /STAT/ CPUT,NFUNC,NGRAD,NFLINE,NLINE,NFL,NWNTN
      REAL NORM
      LOGICAL NEWXW,NEW,LINEAR



      CALL SETUP(F0,X0,M,K,KV,F,NEWXW,GRAD,H,XW,EVAL,E,
     $    DLSMX,NEW,LINEAR,NORM,DIR,EWORK,DELS)



    5 IF (CONVGD(NEWXW)) GO TO 50
C
C    ACCELERATE ALONG THE VALLEY
C
      IF (.NOT. ACCPOS(DIROLD,NEW,KV,K,NORM,DIR,X,XW,
     $    DSBASE,DS12,DS23,DS,NEWXW,M,LINEAR,N)) GO TO 30
   10 CALL UPDS(DSBASE,DS12,DS23,DS,F,F1,F2)
      DO 20 I = 1,M
   20 XW(I) = X(I) + DS * DIR(I)
      CALL FUNC(F,XW,M)
      CALL STORF0(X0,F0,XW,F,M)
      NFL = NFL + 1
      IF (.NOT. FINC(F,F2,LINEAR,NEWXW,XW,NORM,K,KV,EWORK,
     $  E,DELS,DIROLD,DIR,M,DLSMX,DS23,H,GRAD,EVAL,DS12,
     $  DSBASE,F1,X,X0,F0)) GO TO 10
C
C    GET BACK ON THE VALLEY
C
   30 IF (ONVLLY(DLSMX,NEWXW,N,K,M,NEW,LINEAR)) GO TO 40
      CALL GETDIR(N,DIR,K,DELS,E,M)
      IF (KV .GT. K) CALL INCKV(U,K,KV,DELS,E,DIROLD,DIR,M,
     $    DLSMX)
      CALL SEARCH(DIR,XW,F,U,M,NEWXW,K)
```

```
      CALL STORF0(X0,F0,XW,F,M)
      IF (NEWXW) CALL UPGHEV(XW,H,GRAD,M,EVAL,E,EWORK,KV,
     $    DLSMX,K,DELS) GO TO 30
   40 IF (MINHV(K,KV,M,NEWXW,NORM,EWORK,E,DELS,DIROLD,DIR,
     $    F,F0,X0,H,GRAD,EVAL,DLSMX,NEW,LINEAR))
     $    CALL HYPERC(XW,GRAD,M,EVAL,E,KV,K,DLSMX,NEW,
     $       LINEAR,F,F0,X0,NORM,DIR,DELS)
      GO TO 5



   50 RETURN
      END
      LOGICAL FUNCTION BRCKTD(NORM,K,KV,EWORK,E,DELS,
     $    DIROLD,M)
C
C    CHECK TO SEE IF WE ARE DONE ACCELERATING ALONG THE
C    VALLEY.  IF THE NEW DIRECTION IS ESSENTIALLY POINTING
C    180 DEGREES AWAY FROM THE OLD DIRECTION (DIROLD), THEN
C    THE MINIMUM ON THE VALLEY HAS BEEN BRACKETED.
C
      DIMENSION EWORK(M),E(M,M),DIROLD(M),DELS(M)
      REAL NORM
      NORM = 0.0
      DO 10 I = 1,M
   10 EWORK(I) = 0.0
      KP1 = K + 1
      DO 20 J = KP1,KV
      DO 20 I = 1,M
   20 EWORK(I) = EWORK(I) + DELS(J) * E(I,J)
      DO 30 I = 1,M
   30 NORM = NORM + EWORK(I) * EWORK(I)
      NORM = SQRT(NORM)
      BRCKTD = .FALSE.
      IF (NORM .EQ. 0.0) GO TO 60
      DO 40 I = 1,M
   40 EWORK(I) = EWORK(I) / NORM
      DOT = 0.0
      DO 50 I = 1,M
   50 DOT = DOT + DIROLD(I) * EWORK(I)
      IF (DOT .LT. - 0.7) BRCKTD = .TRUE.
   60 RETURN
      END
      SUBROUTINE CLOSIN(F,F1,F2,F3,DSBASE,DS12,DS23,X1,DIR,
     $    XWORK,K,M)
C
C  THIS SUBROUTINE TAKES A THREE POINT PATTERN ON THE
C  MINIMUM AND FINDS THE MINIMUM.  THE MINIMUM IS RETURNED
```

```
C   IN X1, THE OBJECTIVE VALUE AT THE MINIMUM IS RETURNED IN
C   F1.
C
      REAL X1(M), DIR(M), XWORK(M)
      COMMON /IUNIT/ NIN, NOUT
      DSMIN = 0.005
      IF (K.EQ.M) DSMIN = AMIN1((DS12 + DS23) / 100.0,0.005)
      IF (DSMIN .LT. 1.0E-7) DSMIN = 1.0E-7
C
C   CLOSE IN ON THE THREE POINT PATTERN
C
   40 CALL LININT(F1,F2,F3,DS12,DS23,DS1MN)
      IF (ABS(DS12 - DS1MN) .LE. DSMIN) GO TO 80
      IF (DS12*20.0.LT.DS23 .AND. ABS(DS1MN-DS12).LT.DS12)
     1   DS1MN = 2.0 * DS12
      IF (DS23*20.0.LT.DS12 .AND. ABS(DS1MN-DS12).LT.DS23)
     1   DS1MN = DS12 - DS23
      DS = DS1MN + DSBASE
      DO 45 I = 1,M
   45 XWORK(I) = X1(I) + DS * DIR(I)
      CALL FUNC(FMN,XWORK,M)
      IF (DS1MN - DS12 .GE. 0.0) GO TO 60
C
C   CODE FOR (DS1MN .LT. DS12)
C
      IF (FMN - F2 .GE. 0.0) GO TO 50
C
C   (DS1MN .LT. DS12) .AND. (FMN .LT. F2)
C
      DS23 = DS12 - DS1MN
      DS12 = DS1MN
      F3 = F2
      F2 = FMN
      GO TO 40
C
C   (DS1MN .LT. DS12) .AND. (FMN .GE. F2)
C
   50 DSBASE = DSBASE + DS1MN
      DS12 = DS12 - DS1MN
      F1 = FMN
      GO TO 40
C
C   CODE FOR (DS1MN .GT. DS12)
C
   60 IF (FMN - F2 .GE. 0.0) GO TO 70
C
C   (DS1MN .GT. DS12) .AND. (FMN .LT. F2)
C
```

```
      DSBASE = DSBASE + DS12
      DS12 = DS1MN - DS12
      DS23 = DS23 - DS12
      F1 = F2
      F2 = FMN
      GO TO 40
C
C  (DS1MN .GT. DS12) .AND. (FMN .GE. F2)
C
   70 DS23 = DS1MN - DS12
      F3 = FMN
      GO TO 40
C
C  STUFF ANSWER INTO X1 THEN RETURN
C
   80 DS = DSBASE + DS12
      DO 130 I = 1,M
      X1(I) = X1(I) + DS * DIR(I)
  130 CONTINUE
      F = F2
      RETURN
      END
      LOGICAL FUNCTION CONVGD(NEWXW)
C
C CHECK CONVERGENCE OF THE ALGORITHM AT X0.
C
      LOGICAL NEWXW
      CONVGD = .TRUE.
      IF (NEWXW) CONVGD = .FALSE.
      RETURN
      END
      LOGICAL FUNCTION FINC(F,F2,LINEAR,NEWXW,XW,NORM,K,KV,
     $   EWORK,E,DELS,DIROLD,DIR,M,DLSMX,DS23,H,GRAD,EVAL,
     $   DS12,DSBASE,F1,X,X0,F0)
C
C   TEST TO SEE IF F HAS INCREASED WHILE STEPPING TO GET AN
C   INCREASE.
C
      DIMENSION XW(M),EWORK(M),E(M,M),DELS(M),DIROLD(M)
      DIMENSION H(M,M),GRAD(M),EVAL(M),X(M),X0(M),DIR(M)
      REAL NORM
      LOGICAL LINEAR,NEWXW
      FINC = .FALSE.
      IF (F .LE. F2) GO TO 20
      FINC = .TRUE.
C
C   ALGORITHM ISN'T GOING ANY WHERE.
C
```

```
      IF (DS23 .GT. 1.0E-6) GO TO 10
      LINEAR = .TRUE.
      NEWXW = .FALSE.
      GO TO 20
C
C   SEARCH A LINEAR VALLEY OR SET UP TO GET BACK ON THE
C   VALLEY.
C
   10 CALL UPGHEV(XW,H,GRAD,M,EVAL,E,EWORK,KV,DLSMX,K,DELS)
      IF (BRCKTD(NORM,K,KV,EWORK,E,DELS,DIROLD,M).AND.
     $    DSLMX.LT.0.01)
     $   CALL LINERV(LINEAR,DS12,DS23,DSBASE,F1,F2,F,XW,X,
     $      DIR,M,K,KV,H,GRAD,EVAL,E,EWORK,DLSMX,DELS)
      CALL STORF0(X0,F0,XW,F,M)
   20 RETURN
      END
      SUBROUTINE FUNC(F,X,M)
C
C  THIS IS THE FUNCTION TO BE MINIMIZED
C
C  STARTING POINT IS (-1.5,1,1,1).  MINIMUM IS (1,1,1,1).
C
      COMMON /STAT/ CPUT,NFUNC,NGRAD,NFLINE,NLINE,NFL,NWNTN
      REAL X(M)
      NFUNC = NFUNC + 1
      F = 225.0 * (X(4) - X(3) ** 2) ** 2
     1  + 100.0 * (X(3) - X(2) ** 2) ** 2
     2  +  25.0 * (X(2) - X(1) ** 2) ** 2
     3  + (1.0 - X(1)) ** 2
C
C   CONVERGENCE TEST TO CHECK ALGORITHMS.
C
      IF (F .GT. 1.E-13) RETURN
      CPUT = (TIME(12) - CPUT) * 2.4E-6
      WRITE(6,*//) CPUT,NFUNC,NGRAD,N,NFLINE,NLINE,NWNTN,X,F
      STOP
      END
      SUBROUTINE GETDIR(N,DIR,K,DELS,E,M)
C
C   GET THE DIRECTION USED TO GET BACK TO THE VALLEY AND
C   STORE IT IN DIR.  ALSO CALCULATE THE MAXIMUM DELS VALUE
C   WHICH IS USED LATER TO CHECK CONVERGENCE.
C
      DIMENSION DIR(M),DELS(M),E(M,M)
      N = N + 1
      DO 10 I = 1,M
   10 DIR(I) = 0.0
      DO 20 I = 1,K
```

```
      DO 20 J = 1,M
   20 DIR(J) = DIR(J) + DELS(I) * E(J,I)
      RETURN
      END
      SUBROUTINE GRADNT(GRAD,H,X,M)
C
C  THIS SUBROUTINE CALCULATES THE ANALYTICAL GRADIENT OF THE
C  OBJECTIVE FUNCTION
C
      COMMON /STAT/ CPUT,NFUNC,NGRAD,NFLINE,NLINE,NFL,NWNTN
      REAL GRAD(M), X(M), H(M,M)
      NGRAD = NGRAD + 1
      GRAD(4) = 450.0 * (X(4) - X(3) ** 2)
      GRAD(3) = 200.0 * (X(3) - X(2) ** 2)
      GRAD(2) =  50.0 * (X(2) - X(1) ** 2)
      GRAD(1) = - 2.0 * X(1) * GRAD(2) - 2.0 * (1.0 - X(1))
      GRAD(2) = - 2.0 * X(2) * GRAD(3) + GRAD(2)
      GRAD(3) = - 2.0 * X(3) * GRAD(4) + GRAD(3)
      H(1,1) = - 100.0 * X(2) + 300.0 * X(1) ** 2 + 2.0
      H(1,2) = - 100.0 * X(1)
      H(2,1) = H(1,2)
      H(1,3) = 0.0
      H(3,1) = 0.0
      H(1,4) = 0.0
      H(4,1) = 0.0
      H(2,2) = - 400.0 * X(3) + 1200.0 * X(2) ** 2 + 50.0
      H(2,3) = - 400.0 * X(2)
      H(3,2) = H(2,3)
      H(2,4) = 0.0
      H(4,2) = 0.0
      H(3,3) = - 900.0 * X(4) + 2700.0 * X(3) ** 2 + 200.0
      H(3,4) = - 900.0 * X(3)
      H(4,3) = H(3,4)
      H(4,4) = 450.0
      RETURN
      END
      SUBROUTINE HYPERC(XW,GRAD,M,EVAL,E,KV,K,DLSMX,NEW,
     $   LINEAR,F,F0,X0,NORM,DIR,DELS)
C
C   DETERMINE THE EIGEN VECTORS THAT ARE TO BE INCLUDED IN
C   THE HYPER CROSSECTION AND THE EIGEN VECTORS THAT DEFINE
C   THE HYPER VALLEY.  EIGEN VECTORS WITH POSITIVE EIGEN
C   VALUES ARE NOT GROUPED TOGETHER WITH EIGEN VECTORS WITH
C   NEGATIVE EIGEN VALUES IN THE HYPER VALLEY.
C
      DIMENSION XW(M),GRAD(M),EVAL(M),E(M,M),X0(M),DIR(M)
      DIMENSION DELS(M)
      REAL NORM
```

```
      LOGICAL NEW,LINEAR
      K = KV
      IF (KV .EQ. M) GO TO 120
      IF (EVAL(KV+1) .LT. 0.0) GO TO 80
C
C    PUT EIGEN VECTORS WITH POSITIVE EIGEN VALUES IN THE
C    HYPER VALLEY.
C
   30 IF (KV .EQ. M) GO TO 120
      IF (EVAL(KV+1) .LT. 0.0) GO TO 70
      DELSJ = 0.0
      DO 40 I = 1,M
   40 DELSJ = DELSJ + GRAD(I) * E(I,KV+1)
      DELS(KV+1) = - DELSJ / ABS(EVAL(KV+1))
      IF (KV .LE. K) GO TO 50
      IF (EVAL(KV+1) .LT. EVLKP1) GO TO 120
      KV = KV + 1
      GO TO 30
   50 KV = KV + 1
      IF (ABS(DELS(KV)) .LT. 0.01) GO TO 60
      EVLKP1 = EVAL(KV) / 2.0
      GO TO 30
   60 K = KV
      GO TO 30
C
C    DON'T PUT EIGEN VECTORS WITH POSITIVE EIGEN VALUES
C    TOGETHER WITH EIGEN VECTORS WITH NEGATIVE EIGEN VALUES
C    IN THE HYPER VALLEY.
C
   70 IF (KV .GT. K) GO TO 120
C
C    PUT EIGEN VECTORS WITH NEGATIVE EIGEN VALUES IN THE
C    HYPER VALLEY.
C
   80 IF (KV .EQ. M) GO TO 120
      DELSJ = 0.0
      DO 90 I = 1,M
   90 DELSJ = DELSJ + GRAD(I) * E(I,KV+1)
      DELS(KV+1) = - DELSJ / ABS(EVAL(KV+1))
      IF (KV .LE. K) GO TO 100
      IF (ABS(EVAL(KV+1)) .GT. EVLKP1) GO TO 120
      KV = KV + 1
      GO TO 80
  100 KV = KV + 1
      IF (ABS(DELS(KV)) .LT. 0.01) GO TO 110
      EVLKP1 = ABS(EVAL(KV)) * 2.0
      GO TO 80
  110 K = KV
```

```
      GO TO 80
C
C   UPDATE DLSMX, NEW, LINEAR, F, X0, DIR AND NORM
C
  120 DLSMX = 0.0
      DO 130 I = 1,K
  130 IF (ABS(DELS(I)) .GT. DLSMX) DLSMX = ABS(DELS(I))
      NEW = .TRUE.
      LINEAR = .FALSE.
      F = F0
      DO 140 I = 1,M
      DIR(I) = 0.0
  140 XW(I) = X0(I)
      NORM = 0.0
      KP1 = K + 1
      IF (KP1 .GT. M) GO TO 180
      DO 150 J = KP1,KV
      DO 150 I = 1,M
  150 DIR(I) = DIR(I) + DELS(J) * E(I,J)
      DO 160 I = 1,M
  160 NORM = NORM + DIR(I) * DIR(I)
      NORM = SQRT(NORM)
      IF (NORM .EQ. 0.0) GO TO 180
      DO 170 I = 1,M
  170 DIR(I) = DIR(I) / NORM
  180 CONTINUE
      RETURN
      END
      SUBROUTINE INCKV(U,K,KV,DELS,E,DIROLD,DIR,M,DLSMX)
C
C   INCLUDE EIGENVECTOR K IN THE SEARCH TO GET BACK IF IT
C   WILL HELP THE ALGORITHM TO ACCELERATE ALONG THE VALLEY.
C
      DIMENSION U(M),DELS(M),E(M,M),DIROLD(M),DIR(M)
      DO 10 I = 1,M
   10 U(I) = 0.0
      KP1 = K + 1
      DO 20 J = KP1,KV
      DO 20 I = 1,M
   20 U(I) = U(I) + DELS(J) * E(I,J)
      DELSK = 0.0
      DO 30 I = 1,M
   30 DELSK = DELSK + U(I) * U(I)
      DELSK = SQRT(DELSK)
      DOT = 0.0
      DO 40 I = 1,M
   40 DOT = DOT + U(I) * DIROLD(I)
      IF (DOT .LE. 0.0) GO TO 80
```

```
      IF (ABS(DELSK) .GT. 100.0*DLSMX) GO TO 50
      DELSK = 1.0
      GO TO 60
   50 DELSK = DLSMX / DELSK
   60 DO 70 I = 1,M
   70 DIR(I) = DIR(I) + DELSK * U(I)
   80 RETURN
      END
      SUBROUTINE LINERV(LINEAR,DS12,DS23,DSBASE,F1,F2,F,XW,
     $   X,DIR,M,K,KV,H,GRAD,EVAL,E,EWORK,DLSMX,DELS)
C
C    IF THE VALLEY IS ESSENTIALLY STRAIGHT THEN CLOSE IN ON
C    THE THREE POINT PATTERN OBTAINED ALONG THE VALLEY.
C
      DIMENSION XW(M),X(M),DIR(M),GRAD(M),EVAL(M),H(M,M)
      DIMENSION DELS(M),E(M,M),EWORK(M)
      COMMON /STAT/ CPUT,NFUNC,NGRAD,NFLINE,NLINE,NFL,NWNTN
      LOGICAL LINEAR
      LINEAR = .TRUE.
      IF (DS12 .NE. 0.0) GO TO 20
C
C    CALL FUNC TO SET UP A THREE POINT PATTERN IF NEEDED.
C
      DS12 = 1.0E-6
      DS23 = DS23 - DS12
      F1 = F2
      DO 10 I = 1,M
   10 XW(I) = X(I) + DS12 * DIR(I)
      NFLINE = NFLINE + 1
      CALL FUNC(F2,XW,M)
C
C    CLOSE IN ON THE THREE POINT PATTERN.
C
   20 NFLINE = NFLINE + NFL
      IF (F2 .GT. F1) GO TO 40
      F3 = F
      CALL CLOSIN(F,F1,F2,F3,DSBASE,DS12,DS23,X,DIR,XW,K,M)
      DO 30 I = 1,M
   30 XW(I) = X(I)
      CALL UPGHEV(XW,H,GRAD,M,EVAL,E,EWORK,KV,DLSMX,K,DELS)
      GO TO 50
   40 F1 = F
   50 RETURN
      END
      SUBROUTINE LININT(F1,F2,F3,DS12,DS23,DS1MN)
C
C  THIS SUBROUTINE FINDS THE MINIMUM ON A LINE BY QUADRATIC
C  INTERPOLATION GIVEN A THREE POINT PATTERN
```

```
C
      DS13 = DS12 + DS23
      X2SQ = DS12 * DS12
      X3SQ = DS13 * DS13
      DENOM = DS23 * F1 - DS13 * F2 + DS12 * F3
      IF (DENOM .EQ. 0) GO TO 20
      DS1MN = - 0.5 * ((X2SQ - X3SQ) * F1 + X3SQ * F2 -
     1   X2SQ * F3) / DENOM
      GO TO 30
   20 DS1MN = DS12 + DS23 * 0.38
      IF (DS12 .GT. DS23) DS1MN = DS12 * 0.62
   30 RETURN
      END
      LOGICAL FUNCTION MINHV(K,KV,M,NEWXW,NORM,EWORK,E,
     $   DELS,DIROLD,DIR,F,F0,X0,H,GRAD,EVAL,DLSMX,NEW,
     $   LINEAR)
C
C   PERFORM THE CHECKS NEEDED TO SEE IF IT IS POSSIBLE TO
C   SETUP A NEW HYPER VALLEY.
C
      DIMENSION EWORK(M),E(M,M),DELS(M),DIROLD(M),DIR(M)
      DIMENSION GRAD(M),EVAL(M),X0(M),H(M,M)
      LOGICAL NEWXW,NEW,LINEAR
      REAL NORM
      MINHV = .FALSE.
      IF (K.GE.M) GO TO 10
      NEWXW = .TRUE.
      IF (.NOT. BRCKTD(NORM,K,KV,EWORK,E,DELS,DIROLD,M)
     $   .AND. .NOT. LINEAR) GO TO 10
      IF (F .NE. F0) CALL UPGHEV(X0,H,GRAD,M,EVAL,E,EWORK,
     $   KV,DLSMX,K,DELS)
      MINHV = .TRUE.
   10 DO 20 I = 1,M
   20 DIR(I) = EWORK(I)
      NEW = .FALSE.
      RETURN
      END
      LOGICAL FUNCTION ONVLLY(DLSMX,NEWXW,N,K,M,NEW,LINEAR)
C
C   CHECK TO SEE IF CURRENT POINT IS ON THE CURRENT HYPER
C   VALLEY.
C
      LOGICAL NEWXW,NEW,LINEAR
      ONVLLY = .TRUE.
      IF (((DLSMX.GT.0.01.AND.NEWXW.AND.3.GE.N.AND.K.LT.M)
     $   .OR.(N.EQ.1.AND.(.NOT.NEW.OR.K.EQ.M))).AND.
     $   .NOT.LINEAR) ONVLLY = .FALSE.
      RETURN
```

```
      END
      SUBROUTINE SEARCH(DIR,X,F,XWORK,M,NEWXW,K)
C
C  THIS SUBROUTINE PERFORMS A LINEAR SEARCH STARTING AT X
C  IN THE DIRECTION DIR.  THE MINIMUM ON THE LINE IS
C  RETURNED IN X. THE FUNCTION VALUE IS RETURNED IN F.
C
      REAL DIR(M), X(M), XWORK(M)
      LOGICAL NEWXW
      COMMON /STAT/ CPUT,NFUNC,NGRAD,NFLINE,NLINE,NFL,NWNTN
      IF (K .EQ. M) NWNTN = NWNTN + 1
      NLINE = NLINE + 1
      NEWXW = .FALSE.
      DSMIN = 1.0E-7
      DS = 0.0
      DO 5 I = 1,M
    5 DS = DS + DIR(I) * DIR(I)
      DS = SQRT(DS)
      IF (DS .EQ. 0.0) GO TO 90
      DO 6 I = 1,M
    6 DIR(I) = DIR(I) / DS
      IF (DS .GT. 0.1) DS = SQRT(0.1 * DS)
      DSBASE = 0.0
      F1 = F
C
C  STEP FROM F1 ALONG DIR TO F2
C
   10 DO 20 I = 1,M
      XWORK(I) = X(I) + DS * DIR(I)
   20 CONTINUE
      NFLINE = NFLINE + 1
      CALL FUNC(F2,XWORK,M)
      DS12 = DS
      IF (F2 .LT. F1 .AND. DS .LE. 0.1) GO TO 55
      IF (F2 .GT. F1) GO TO 60
C
C  STEP FROM F2 ALONG DIR TO F3
C
   30 DS23 = 2.0 * DS12
      DS = DS12 + DS23 + DSBASE
      DO 40 I = 1,M
      XWORK(I) = X(I) + DS * DIR(I)
   40 CONTINUE
      NFLINE = NFLINE + 1
      CALL FUNC(F3,XWORK,M)
      IF (F3 .GE. F2) GO TO 80
C
C  SHIFT F2 TO F1 AND F3 TO F2
```

```
C
      DSBASE = DSBASE + DS12
      DS12 = DS23
      F1 = F2
      F2 = F3
      GO TO 30
C
C  NEWTON STEPS
C
   55 DO 56 I = 1,M
   56 X(I) = XWORK(I)
      F = F2
      NEWXW = .TRUE.
      GO TO 90
C
C  DS IS TOO LARGE SO CLOSE IN
C
   60 DS12 = DSMIN
      F3 = F2
      DS23 = DS - DS12
      DS = DS12
      DO 70 I = 1,M
      XWORK(I) = X(I) + DS * DIR(I)
   70 CONTINUE
      NFLINE = NFLINE + 1
      CALL FUNC(F2,XWORK,M)
      IF (F2 .GT. F1) GO TO 90
C
C  THREE POINT PATTERN HAS BEEN OBTAINED ON THE MINIMUM
C
   80 CALL CLOSIN(F,F1,F2,F3,DSBASE,DS12,DS23,X,DIR,XWORK,
     $   K,M)
      NEWXW = .TRUE.
   90 CONTINUE
      RETURN
      END
      SUBROUTINE SETUP(F0,X0,M,K,KV,F,NEWXW,GRAD,H,XW,EVAL,
     $   E,DLSMX,NEW,LINEAR,NORM,DIR,EWORK,DELS)
C
C   INITIALIZE ALL THE VARIABLES NEEDED TO START THE
C   ALGORITHM.
C
      DIMENSION X0(M),GRAD(M),XW(M),EVAL(M),E(M,M),DIR(M)
      DIMENSION DELS(M),H(M,M),EWORK(M)
      REAL NORM
      LOGICAL NEWXW,NEW,LINEAR
      CALL FUNC(F0,X0,M)
      K = 0
```

```
      KV = 0
      F = F0
      NEWXW = .TRUE.
      CALL GRADNT(GRAD,H,X0,M)
      CALL EIGRS(H,M,11,EVAL,E,M,EWORK,IER)
      CALL HYPERC(XW,GRAD,M,EVAL,E,KV,K,DLSMX,NEW,LINEAR,F,
     $   F0,X0,NORM,DIR,DELS)
C
C   IF K = 0 THE STARTING POINT IS NOT ON A HYPER VALLEY.
C
      IF (K .NE. 0) NEW = .FALSE.
      IF (K .EQ. 0) CALL HYPERC(XW,GRAD,M,EVAL,E,KV,K,DLSMX,
     $   NEW,LINEAR,F,F0,X0,NORM,DIR,DELS)
      RETURN
      END
      SUBROUTINE STORF0(X0,F0,X,F,M)
C
C   ALWAYS STORE THE LOWEST FUNCTION VALUE AND THE
C   ASSOCIATED SOLUTION VECTOR IN F0 AND X0.
C
      DIMENSION X0(M),X(M)
      IF (F .GE. F0) GO TO 20
      F0 = F
      DO 10 I = 1,M
   10 X0(I) = X(I)
   20 RETURN
      END
      SUBROUTINE UPDS(DSBASE,DS12,DS23,DS,F,F1,F2)
C
C   CALCULATE DS AND PERFORM THE CALCULATIONS NEEDED TO
C   SETUP PROCEDURE CLOSEIN FOR A LINEAR VALLEY.
C
      DSBASE = DSBASE + DS12
      DS12 = DS23
      DS23 = DS23 * 2.0
      IF (DS23 .EQ. 0.0) DS23 = DS
      F1 = F2
      F2 = F
      DS = DSBASE + DS12 + DS23
      RETURN
      END
      SUBROUTINE UPGHEV(X,H,GRAD,M,EVAL,E,EWORK,KV,DLSMX,K,
     $   DELS)
C
C   GET THE GRADIEND, HESSIAN, EIGENVECTORS AND DELS AT THE
C   POINT X.  DELS IS THE NEWTON STEP VECTOR IN THE EIGEN
C   VECTOR SPACE.
C
```

```
      DIMENSION X(M),H(M,M),GRAD(M),EVAL(M),E(M,M),EWORK(M)
      CALL GRADNT(GRAD,H,X,M),DELS(M)
      CALL EIGRS(H,M,11,EVAL,E,M,EWORK,IER)
      DO 20 J = 1,KV
      DELSJ = 0.0
      DO 10 I = 1,M
   10 DELSJ = DELSJ + GRAD(I) * E(I,J)
      DELS(J) = - DELSJ / ABS(EVAL(J))
   20 CONTINUE
      DLSMX = 0.0
      DO 30 I = 1,K
   30 IF (ABS(DELS(I)) .GT. DLSMX) DLSMX = ABS(DELS(I))
      RETURN
      END
```

APPENDIX C

Listing of ALGOL Version of BANANA

```
BEGIN
FILE            CARD(BUFFERS=2, MAXRECSIZE=14, MYUSE=1),
                LINE(BUFFERS=2, MAXRECSIZE=22, MYUSE=2,
                 KIND=PRINTER),
                REMOT(KIND=REMOTE, MYUSE=IO, MAXRECSIZE=22);


INTEGER         M,NDEF,NFL,NFLINE,NFUNC,NGRAD,NHV,NLINE,WNTN;
REAL            CPUT,F0;
REAL ARRAY      X0[0:2];
LABEL           EOP;


PROCEDURE BANANA(X0,F0,M); INTEGER M; REAL F0;
                           REAL ARRAY X0[0];
BEGIN
INTEGER         I,K,KVALLEY,N,ND,NEG,NZRO;
REAL            DS,DSBASE,DLSMX,DS12,DS23,EVALKP1,F,F1,F2,F3,
                NORM;
BOOLEAN         LINEAR,NEW,NEWXWORK;
REAL ARRAY      DELS,DIR,DIROLD,EVAL,EWORK,GRAD,U,X,
                XWORK[0:M];
REAL ARRAY      E[0:M,0:M];
REAL ARRAY      H[0:M*(M+1)/2];
DEFINE          LOOP(I) = FOR I := 1 STEP 1 UNTIL M DO #;
DEFINE          NOTONVALLEY = ((DLSMX > 0.01 AND NEWXWORK
                   AND 3 GEQ N AND K < M) OR (N = 1 AND
                   (NOT NEW OR K = M))) AND NOT LINEAR #;
PROCEDURE FUNC(X,F); REAL ARRAY X[0]; REAL F; FORWARD;
PROCEDURE LINEARVALLEY; FORWARD;
REAL PROCEDURE LININT; FORWARD;
PROCEDURE POWER(EVAL,V,N);   VALUE N; INTEGER N; REAL EVAL;
                             REAL ARRAY V[0]; FORWARD;
PROCEDURE STOREF0; FORWARD;
PROCEDURE UPDATEGRADHESSIANEIGENVECTORS(X); REAL ARRAY X[0];
                                            FORWARD;


%
%     INPUT     DESCRIPTION
%     X0        INITIAL GUESS FOR THE OPTIMIZATION
%     M         NUMBER OF OPTIMIZING VARIABLES
%
%     OUTPUT    DESCRIPTION
```

```
%      X0          SOLUTION VECTOR AT THE MINIMUM
%      F0          OBJECTIVE FUNCTION VALUE AT THE MINIMUM
%
```

```
BOOLEAN PROCEDURE ACCELERATIONPOSSIBLE;
BEGIN
INTEGER   I;
      %
      % PERFORM CALCULATIONS NEEDED TO SET UP THE
      % ACCELERATION ALONG THE VALLEY
      %
      LOOP(I) DIROLD[I] := 0.0;
      ACCELERATIONPOSSIBLE := FALSE;
      IF NOT NEW AND KVALLEY > K AND NORM NEQ 0.0 THEN
      BEGIN     % SET UP FOR ACCELERATION
           LOOP(I) DIROLD[I] := DIR[I];
           LOOP(I) X[I] := XWORK[I];
           DSBASE := 0.0;
           DS12 := 0.0;
           DS23 := 0.0;
           IF NORM > 0.1 THEN DS := SQRT(0.1*NORM)
              ELSE DS := NORM;
           NFL := 0;
           ACCELERATIONPOSSIBLE := TRUE;
      END;
      NEWXWORK := TRUE;
      LINEAR := FALSE;
      N := 1;
END OF PROCEDURE ACCELERATIONPOSSIBLE;


BOOLEAN PROCEDURE BRACKETED;
BEGIN
INTEGER   I,J;
REAL      DOT;
      %
      % CHECK TO SEE IF WE ARE DONE ACCELERATING ALONG THE
      % VALLEY.  IF THE NEW DIRECTION IS ESSENTIALLY POINTING
      % 180 DEGREES AWAY FROM THE OLD DIRECTION (DIROLD),
      % THEN THE MINIMUM ON THE VALLEY HAS BEEN BRACKETED.
      %
      NORM := 0.0;
      LOOP(I) EWORK[I] := 0.0;
      FOR J := K+1 STEP 1 UNTIL KVALLEY DO
```

```
      LOOP(I) EWORK[I] := EWORK[I] + DELS[J] * E[J,I];
      LOOP(I) NORM := NORM + EWORK[I] * EWORK[I];
      NORM := SQRT(NORM);
      IF NORM = 0.0 THEN BRACKETED := FALSE ELSE
      BEGIN
           LOOP(I) EWORK[I] := EWORK[I] / NORM;
           DOT := 0.0;
           LOOP(I) DOT := DOT + DIROLD[I] * EWORK[I];
           BRACKETED := DOT < - 0.7;
      END;
END OF BOOLEAN PROCEDURE BRACKETED;
PROCEDURE CLOSEIN(X,F);   REAL F;   REAL ARRAY X[0];
BEGIN
REAL      DS1MN,FMN;
INTEGER   I;
REAL      EPS;
      %
      % THIS PROCEDURE REFINES THE ESTIMATE OF THE MINIMUM
      % ON A LINE SEARCH AFTER THE THREE POINT PATTERN HAS
      % BEEN OBTAINED
      %
      EPS := .005;
      IF K = M THEN EPS := MIN((DS12+DS23)/100.,0.005);
      IF EPS < 1@-7 THEN EPS := 1@-7;
      WHILE ABS(DS12 - DS1MN := LININT) > EPS DO
      BEGIN
           IF DS12*20.0<DS23 AND ABS(DS1MN-DS12)<DS12 THEN
              DS1MN := 2.0 * DS12;
           IF DS23*20.0<DS12 AND ABS(DS1MN-DS12)<DS23 THEN
              DS1MN := DS12 - DS23;
           DS := DS1MN + DSBASE;
           LOOP(I) U[I] := X[I] + DS * DIR[I];
           NFLINE := NFLINE + 1;
           FUNC(U,FMN);
           IF DS1MN > DS12 THEN
           BEGIN
                IF FMN > F2 THEN
                BEGIN
                     DS23 := DS1MN - DS12;
                     F3 := FMN;
                END ELSE
                BEGIN
                     DSBASE := DSBASE + DS12;
                     DS12 := DS1MN - DS12;
                     DS23 := DS23 - DS12;
                     F1 := F2;
                     F2 := FMN;
                END;
```

```
          END ELSE
          BEGIN
               IF FMN > F2 THEN
               BEGIN
                    DSBASE := DSBASE + DS1MN;
                    DS12 := DS12 - DS1MN;
                    F1 := FMN;
               END ELSE
               BEGIN
                    DS23 := DS12 - DS1MN;
                    DS12 := DS1MN;
                    F3 := F2;
                    F2 := FMN;
               END;
          END;
     END;
     DS := DSBASE + DS12;
     F := F2;
     LOOP(I) X[I] := X[I] + DS * DIR[I];
END OF PROCEDURE CLOSEIN;


BOOLEAN PROCEDURE CONVERGED;
BEGIN
     %
     % CHECK CONVERGENCE OF THE ALCORITHM AT X0
     %
     CONVERGED := NOT NEWXWORK;
END OF BOOLEAN PROCEDURE CONVERGED;


PROCEDURE DEFLATE(EVAL,V);   REAL EVAL;   REAL ARRAY V[0];
BEGIN
INTEGER   I,J,L;
     %
     % TRANSFORM THE HESSIAN MATRIX, H, SO THAT THE EIGEN
     % VALUE OF EIGEN VECTOR V IS ZERO.  THIS EXCLUDES V
     % FROM INCLUSION IN THE POWER METHOD OF OBTAINING
     % EIGEN VECTORS.
     %
     NDEF := NDEF + 1;
     LOOP(I) U[I] := EVAL * V[I];
     L := 0;
     LOOP(J)
     FOR I := J STEP 1 UNTIL M DO
     BEGIN     % SUBTRACT OFF A RANK 1 MATRIX
          L := L + 1;
          H[L] := H[L] - U[I] * V[J];
```

```
     END;
END OF PROCEDURE DEFLATE;


BOOLEAN PROCEDURE FINCREASE;
BEGIN
     %
     % TEST TO SEE IF F HAS INCREASED WHILE STEPPING TO GET
     % AN INCREASE.
     %
     FINCREASE := FALSE;
     IF F > F2 THEN IF DS23 LEQ 1.0@-6 THEN
     BEGIN     % ALGORITHM IS NOT GOING ANY WHERE
          LINEAR := TRUE;
          FINCREASE := TRUE;
          NEWXWORK := FALSE;
     END ELSE
     BEGIN     % SEARCH LINEAR VALLEY OR SET UP TO GET BACK
          UPDATEGRADHESSIANEIGENVECTORS(XWORK);
          IF BRACKETED AND DLSMX < 0.01 THEN LINEARVALLEY;
          STOREF0;
          FINCREASE := TRUE;
     END;
END OF PROCEDURE FINCREASE;


PROCEDURE FUNC(X,F); REAL ARRAY X[0]; REAL F;
BEGIN
INTEGER   I;
     %
     % CALCULATE THE OBJECTIVE FUNCTION VALUE USING X.
     % RETURN THE OBJECTIVE FUNCTION VALUE IN F.
     %
     NFUNC := NFUNC + 1;
     IF F < 1.0@-13 THEN
     BEGIN     % CONVERGENCE CHECK TO TEST ALGORITHMS
          GO TO EOP;
     END;
END OF PROCEDURE FUNC;



PROCEDURE GETDIR;
BEGIN
INTEGER   I,J;
     %
     % GET THE DIRECTION USED TO GET BACK TO THE VALLEY AND
```

```
      % STORE IN DIR.  ALSO CALCULATE THE MAXIMUM DELS VALUE
      % WHICH IS USED LATER TO CHECK CONVERGENCE.
      %
      N := N + 1;
      LOOP(I) DIR[I] := 0.0;
      FOR I := 1 STEP 1 UNTIL K DO
         LOOP(J) DIR[J] := DIR[J] + DELS[I] * E[I,J];
END OF PROCEDURE GETDIR;
INTEGER PROCEDURE GETEIGENVECTOR(L);      INTEGER      L;
BEGIN
INTEGER    I,J,LUP,LOW,N;
REAL       EVALUE;
      %
      % IT IS POSSIBLE TO HAVE A NEGATIVE EIGEN VALUE THAT
      % IS LARGER IN ABSOLUTE VALUE THAN THE EIGEN VALUE
      % OF THE CURRENT EIGEN VECTOR THAT IS REQUIRED.  THE
      % POWER METHOD WILL PICK THIS EIGEN VECTOR UP FIRST.
      % THESE EIGEN VECTORS ARE STORED AT THE END OF THE
      % ARRAY E.  THIS PROCEDURE DETERMINES IF THE NEXT
      % EIGEN VECTOR OBTAINED BY THE POWER METHOD HAS A
      % NEGATIVE EIGEN VALUE.  THIS IS DONE BY USING A
      % VECTOR OF ALL ONES AND PERFORMING ONE POWER
      % ITERATION ON A SINGLE ROW OF THE HESSIAN.  BY USING
      % ALL ONES, NO MULTIPLICATIONS ARE REQUIRED.  THE
      % PROPER INDEX IS RETURNED SO THAT THE CORRECT EIGEN
      % VECTOR IS USED AS THE INITIAL GUESS.
      %
      IF EVAL[ND] GEQ 0.0 OR L = ND THEN GETEIGENVECTOR := L
         ELSE
      BEGIN
           EVALUE := 0.0;
           J := 0;
           LUP := 0;
           WHILE EVALUE = 0.0 AND M GEQ J := J + 1 DO
           BEGIN      % ONE POWER ITERATION ON ONE ROW
                FOR I := J STEP 1 UNTIL M DO
                      % DIAGONAL AND UPPER HALF
                   EVALUE := EVALUE + H[LUP:=LUP+1];
                LOW := 0;
                N := M;
                FOR I := J STEP -1 UNTIL 2 DO
                BEGIN      % LOWER HALF OF THE HESSIAN
                     EVALUE := EVALUE + H[LOW+J];
                     LOW := LOW + N;
                     N := N - 1;
                END;
           END;
           IF EVALUE < 0 THEN
```

```
               GETEIGENVECTOR := ND ELSE
               GETEIGENVECTOR := L;
     END;
END OF PROCEDURE GETEIGENVECTOR;


PROCEDURE GETGRAD(GRAD,H,X); REAL ARRAY GRAD,X[0],H[0];
BEGIN
     %
     % PROCEDURE THAT USES X TO GET THE GRADIENT (GRAD) AND
     % HESSIAN (H) OF THE OBJECTIVE FUNCTION
     %
     NGRAD := NGRAD + 1;
END OF PROCEDURE GETGRAD;


PROCEDURE HV(V);   REAL ARRAY V[0];
BEGIN
INTEGER   I,J,L;
     %
     % THIS IS A UTILITY PROCEDURE USED IN THE PROCEDURE
     % POWER TO OBTAIN THE PRODUCT H * V
     %
     NHV := NHV + 1;
     L := 0;
     LOOP(J)
     BEGIN     % DIAGONAL AND UPPER HALF OF THE HESSIAN
          U[J] := 0.0;
          FOR I := J STEP 1 UNTIL M DO
          U[J] := U[J] + H[L:=L+1] * V[I];
     END;
     L := 0;
     FOR J := 1 STEP 1 UNTIL M - 1 DO
     BEGIN     % LOWER HALF OF THE MATRIX
          L := L + 1;
          FOR I := J + 1 STEP 1 UNTIL M DO
          U[I] := U[I] + H[L:=L+1] * V[J];
     END;
     LOOP(I) V[I] := U[I];
END OF PROCEDURE HV;


PROCEDURE HYPERCROSSECTION;
BEGIN
INTEGER   I,J,L,N;
REAL      DELSL,EVALN;
     %
     % DETERMINE THE EIGEN VECTORS THAT ARE TO BE INCLUDED
```

```
% IN THE HYPERCROSSECTION AND THE EIGEN VECTORS OF THE
% VALLEY
%
N := MAX(K,KVALLEY);
L := N;
NZRO := 0;
NEG := 0;
WHILE ND > L AND (KVALLEY LEQ K OR ABS(EVAL[L]) GEQ
   EVALKP1) DO
BEGIN % DO POWER ITERS AND SET UP POSITIVE EIGEN VALUES
     IF N NEQ 0 THEN DEFLATE(EVAL[N],E[N,*]);
     LOOP(I) EWORK[I] := - GRAD[I];
     POWER(EVALN,EWORK,100);
     IF EVALN = 0.0 THEN
     BEGIN     % ZERO EIGEN VALUE
          I := ND;
          ND := L;
          J := L;
          WHILE M LEQ I := I + 1 DO
          BEGIN     % SHIFT NEGATIVE EIGEN VECTORS UP
               J := J + 1;
               DELS[J] := DELS[I];
               EVAL[J] := EVAL[I];
               LOOP(N) E[J,N] := E[I,N];
          END;
          WHILE M LEQ J := J + 1 DO
          BEGIN     % ZERO OUT REMAINING VALUES
               NZRO := NZRO + 1;
               DELS[J] := 0.0;
               EVAL[J] := 0.0;
               LOOP(N) E[J,N] := 0.0;
          END;
          N := 0;
     END ELSE
     IF EVALN > 0.0 THEN
     BEGIN     % POSITIVE EIGEN VALUE
          L := L + 1;
          N := L;
          DELSL := 0.0;
          LOOP(I) DELSL := DELSL + EWORK[I] * GRAD[I];
          DELS[L] := - DELSL / ABS(EVALN);
          IF KVALLEY LEQ K AND ABS(DELS[L]) < 0.01 THEN
             K := L ELSE
          IF KVALLEY LEQ K THEN
          BEGIN
               EVALKP1 := EVALN / 2.0;
               KVALLEY := L;
          END ELSE
```

```
                IF EVALN GEQ EVALKP1 THEN KVALLEY := L;
            END ELSE
            BEGIN     % NEGATIVE EIGEN VALUE
                NEG := NEG + 1;
                N := ND;
                ND := ND - 1;
            END;
            EVAL[N] := EVALN;
            LOOP(I) E[N,I] := EWORK[I];
        END;
        L := ND;
        IF KVALLEY LEQ K THEN
        WHILE M GEQ L := L + 1 AND (KVALLEY LEQ K OR
           ABS(EVAL[L-1]) LEQ EVALKP1) DO
        IF EVAL[L] = 0.0 THEN KVALLEY := L := M ELSE
        BEGIN % ADD IN EIGEN VECTORS WITH NEGATIVE EIGEN VALUES
            DELSL := 0.0;
            LOOP(I) DELSL := DELSL + E[L,I] * GRAD[I];
            DELS[L] := - DELSL / ABS(EVAL[L]);
            IF KVALLEY LEQ K AND ABS(DELS[L]) < 0.01 THEN
               K := L ELSE
            IF KVALLEY LEQ K THEN
            BEGIN
                EVALKP1 := ABS(EVAL[L]) * 2.0;
                KVALLEY := L;
            END ELSE
            IF ABS(EVAL[L]) LEQ EVALKP1 THEN KVALLEY := L;
        END;
        NEW := TRUE;
        LINEAR := FALSE;
        DLSMX := 0.0;
        F := F0;
        LOOP(I) XWORK[I] := X0[I];
        FOR I := 1 STEP 1 UNTIL K DO
           IF ABS(DELS[I]) > DLSMX THEN DLSMX := ABS(DELS[I]);
        NORM := 0.0;
        LOOP(I) DIR[I] := 0.0;
        FOR L := K + 1 STEP 1 UNTIL KVALLEY DO
           LOOP(I) DIR[I] := DIR[I] + DELS[L] * E[L,I];
        LOOP(I) NORM := NORM + DIR[I] * DIR[I];
        NORM := SQRT(NORM);
        IF NORM NEQ 0.0 THEN LOOP(I) DIR[I] := DIR[I] / NORM;
    END OF PROCEDURE HYPERCROSSECTION;


    PROCEDURE INCLUDEKVALLEY;
    BEGIN
    INTEGER   I,J;
```

```
REAL      DELSK,DOT;
     %
     % INCLUDE EIGENVECTOR K IN THE SEARCH TO GET BACK IF
     % IT WILL HELP THE ALGORITHM TO ACCELERATE ALONG THE
     % VALLEY.
     %
     LOOP(I) U[I] := 0.0;
     FOR J := K + 1 STEP 1 UNTIL KVALLEY DO
     LOOP(I) U[I] := U[I] + DELS[J] * E[J,I];
     DELSK := 0.0;
     LOOP(I) DELSK := DELSK + U[I] * U[I];
     DELSK := SQRT(DELSK);
     DOT := 0.0;
     LOOP(I) DOT := DOT + U[I] * DIROLD[I];
     IF ABS(DELSK) > 100.0 * DLSMX THEN
        DELSK := DLSMX / DELSK ELSE DELSK := 1.0;
     IF DOT > 0.0 THEN
        LOOP(I) DIR[I] := DIR[I] + DELSK * U[I];
END OF PROCEDURE INCLUDEKVALLEY;


PROCEDURE LINEARVALLEY;
BEGIN
INTEGER   I;
     %
     % IF THE VALLEY IS ESSENTIALLY STRAIGHT THEN CLOSE IN
     % ON THE THREE POINT PATTERN
     %
     LINEAR := TRUE;
     IF DS12 = 0.0 THEN
     BEGIN     % CALL FUNC TO SET UP THE THREE POINT PATTERN
          DS12 := 1.0@-6;
          DS23 := DS23 - DS12;
          F1 := F2;
          LOOP(I) XWORK[I] := X[I] + DS12 * DIR[I];
          NFLINE := NFLINE + 1;
          FUNC(XWORK,F2);
     END;
     NFLINE := NFLINE + NFL;
     IF F2 LEQ F1 THEN
     BEGIN     % CLOSE IN ON THE THREE POINT PATTERN
          F3 := F;
          CLOSEIN(X,F);
          LOOP(I) XWORK[I] := X[I];
          UPDATEGRADHESSIANEIGENVECTORS(XWORK);
     END ELSE F := F1;
END OF PROCEDURE LINEARVALLEY;
```

```
PROCEDURE LINESEARCH(X,F); REAL ARRAY X[0]; REAL F;
BEGIN
INTEGER   I;
REAL      NORM;
LABEL     OUT;
     %
     % PERFORM A LINE SEARCH IN THE DIRECTION DIR STARTING
     % AT X AND F.  RETURN THE MINIMUM IN X AND F.
     %
     IF K = M THEN WNTN := WNTN + 1;
     NLINE := NLINE + 1;
     NEWXWORK := FALSE;
     NORM := 0;
     LOOP(I) NORM := NORM + DIR[I] * DIR[I];
     NORM := SQRT(NORM);
     IF NORM = 0.0 THEN GO OUT;
     LOOP(I) DIR[I] := DIR[I] / NORM;
     IF NORM > 0.1 THEN DS := SQRT(0.1*NORM) ELSE
         DS := NORM;
     F2 := F;
     DSBASE := 0.0;
     LOOP(I) U[I] := X[I] + DS * DIR[I];
     NFLINE := NFLINE + 1;
     FUNC(U,F3);
     DS12 := 0.0;
     DS23 := DS;
     IF F3 > F2 THEN
     BEGIN      % DS IS LARGE ENOUGH THAT WE HAVE ALREADY
                % GONE UP HILL
          IF DS23 LEQ 1.0@-6 THEN GO OUT;
          F1 := F2;
          DS12 := 1.0@-6;
          DS23 := DS23 - DS12;
          LOOP(I) U[I] := X[I] + DS12 * DIR[I];
          NFLINE := NFLINE + 1;
          FUNC(U,F2);
          IF F2 > F1 THEN GO OUT;
     END ELSE
     IF F3 < F2 AND DS LEQ 0.1 THEN
     BEGIN
          LOOP(I) X[I] := U[I];
          F := F3;
          NEWXWORK := TRUE;
          GO OUT;
     END ELSE
     DO BEGIN      % STEP OUT UNTIL A THREE POINT PATTERN
                   % IS OBTAINED
```

```
          F1 := F2;
          F2 := F3;
          DSBASE := DSBASE + DS12;
          DS12 := DS23;
          DS23 := 2.0 * DS12;
          DS := DSBASE + DS12 + DS23;
          LOOP(I) U[I] := X[I] + DS * DIR[I];
          NFLINE := NFLINE + 1;
          FUNC(U,F3);
     END UNTIL F2 < F3;
     CLOSEIN(X,F);
     NEWXWORK := TRUE;
OUT:
END OF PROCEDURE LINESEARCH;


REAL PROCEDURE LININT;
BEGIN
REAL      DENOM,DS13,X2SQ,X3SQ;
     %
     % THIS PROCEDURE ESTIMATES THE MINIMUM ON A LINE BY
     % QUADRATIC INTERPOLATION GIVEN A THREE POINT PATTERN
     %
     DS13 := DS12 + DS23;
     X2SQ := DS12 * DS12;
     X3SQ := DS13 * DS13;
     DENOM := DS23 * F1 - DS13 * F2 + DS12 * F3;
     IF DENOM NEQ 0 THEN
     LININT := - 0.5 * ((X2SQ - X3SQ) * F1 + X3SQ * F2 -
        X2SQ * F3) / DENOM ELSE
     IF DS12 > DS23 THEN
     LININT := DS12 * 0.62 ELSE
     LININT := DS12 + DS23 * 0.38;
END OF PROCEDURE LININT;


BOOLEAN PROCEDURE MINONHYPERVALLEY;
BEGIN
INTEGER   I;
     %
     % PERFORM THE CHECKS NEEDED TO SEE IF IT IS POSSIBLE
     % TO SETUP A NEW HYPERVALLEY
     %
     MINONHYPERVALLEY := FALSE;
     IF K < M THEN
     BEGIN
          NEWXWORK := TRUE;
          IF BRACKETED OR LINEAR THEN
```

```
          BEGIN
               IF F NEQ F0 THEN
                 UPDATEGRADHESSIANEIGENVECTORS(X0);
               K := KVALLEY;
               MINONHYPERVALLEY := TRUE;
          END;
     END;
     LOOP(I) DIR[I] := EWORK[I];
     NEW := FALSE;
END OF PROCEDURE MINONHYPERVALLEY;


PROCEDURE POWER(EVAL,V,N); VALUE N; INTEGER N; REAL EVAL;
                           REAL ARRAY V[0];
BEGIN
INTEGER   I,J,L;
REAL      VMAX;
     %
     % PERFORM THE POWER METHOD OF GETTING AN EIGEN VECTOR
     % AND AN EIGEN VALUE OF THE HESSIAN MATRIX.  TWO
     % ITERATIONS MUST ALWAYS BE PERFORMED, SO N MUST BE
     % GREATER THEN ONE.
     %
     HV(V);
     VMAX := 0.0;
     LOOP(J) IF ABS(V[J]) > VMAX THEN VMAX := ABS(V[J]);
     IF VMAX = 0.0 THEN
     BEGIN     % SET INITIAL GUESS TO ALL ONES AND TRY AGAIN
          L := 0;      % DIAGONAL AND UPPER HALF OF HESSIAN
          LOOP(J) FOR I := J STEP 1 UNTIL M DO
             V[J] := V[J] + H[L:=L+1];
          L := 0;
          FOR J := 1 STEP 1 UNTIL M - 1 DO
          BEGIN     % LOWER HALF OF THE HESSIAN
               L := L + 1;
               FOR I := J+1 STEP 1 UNTIL M DO
                  V[I] := V[I] + H[L:=L+1];
          END;
          LOOP(J) IF ABS(V[J])>VMAX THEN VMAX := ABS(V[J]);
     END;
     IF VMAX = 0.0 THEN EVAL := 0.0 ELSE
     BEGIN     % PERFORM THE POWER ITERATIONS
          FOR I := 1 STEP 1 UNTIL N - 1 DO
          BEGIN
               VMAX := 0.0;
               LOOP(J) IF ABS(V[J]) > VMAX THEN
                  VMAX := ABS(V[L:=J]);
               LOOP(J) V[J] := V[J] / VMAX;
```

```
                VMAX := V[L];
                HV(V);
            END;
            EVAL := VMAX * V[L];
            VMAX := 0.0;
            LOOP(I) VMAX := VMAX + V[I] * V[I];
            VMAX := SQRT(VMAX);
            LOOP(I) V[I] := V[I] / VMAX;
      END;
END OF PROCEDURE POWER;


PROCEDURE SETUP;
BEGIN
INTEGER    I;
REAL       DELSL;
      %
      % INITIALIZE ALL THE VARIABLES NEEDED TO START THE
      % ALGORITHM.
      %
      FUNC(X0,F0);
      K := 0;
      KVALLEY := 0;
      ND := M;
      F := F0;
      NEG := 0;
      NZRO := 0;
      NEWXWORK := TRUE;
      GETGRAD(GRAD,H,X0);
      HYPERCROSSECTION;
      IF K = 0 THEN
      BEGIN      % STARTING POINT IS NOT ON A VALLEY
            K := KVALLEY;
            IF K < M THEN
            BEGIN      % IF K = M THEN DO NEWTON ITERATIONS
                  KVALLEY := K + 1;
                  IF EVAL[KVALLEY] < 0.0 THEN
                  BEGIN
                        DELSL := 0.0;
                        LOOP(I) DELSL := DELSL + E[KVALLEY,I] *
                           GRAD[I];
                        DELS[KVALLEY] := - DELSL /
                           ABS(EVAL[KVALLEY]);
                  END;
                  IF ABS(DELS[KVALLEY]) < 0.01 THEN
                     K := KVALLEY ELSE
                  IF EVAL[KVALLEY] > 0.0 THEN
                     EVALKP1 := ABS(EVAL[KVALLEY]) / 2.0 ELSE
```

```
                    EVALKP1 := ABS(EVAL[KVALLEY]) * 2.0;
                IF K < M THEN HYPERCROSSECTION;
          END;
     END ELSE NEW := FALSE;
END OF PROCEDURE SETUP;


PROCEDURE STOREF0;
BEGIN
INTEGER   I;
     %
     % ALWAYS STORE THE LOWEST FUNCTION VALUE AND THE
     % ASSOCIATED SOLUTION VECTOR IN F0 AND X0
     %
     IF F < F0 THEN
     BEGIN
          F0 := F;
          LOOP(I) X0[I] := XWORK[I];
     END;
END OF PROCEDURE STOREF0;


PROCEDURE UPDATEDS;
BEGIN
     %
     % CALCULATE DS AND PERFORM THE CALCULATIONS NEEDED TO
     % SETUP PROCEDURE CLOSEIN FOR A LINEAR VALLEY
     %
     DSBASE := DSBASE + DS12;
     DS12 := DS23;
     IF DS23 = 0.0 THEN DS23 := DS ELSE DS23 := DS23 * 2;
     F1 := F2;
     F2 := F;
     DS := DSBASE + DS12 + DS23;
END OF PROCEDURE UPDATEDS;


PROCEDURE UPDATEGRADHESSIANEIGENVECTORS(X); REAL ARRAY X[0];
BEGIN
INTEGER   I,J,L,N;
REAL      DELSI,EVALN;
     %
     % GET THE GRADIENT, HESSIAN, EIGENVECTORS AND DELS AT
     % X.  DELS IS THE NEWTON STEP VECTOR IN THE EIGEN
     % VECTOR SPACE.
     %
     GETGRAD(GRAD,H,X);
     ND := M;
```

```
I := ND - NZRO;
IF I < M THEN
WHILE NEG > 0 DO
BEGIN     % PUT NEGATIVE EIGEN VECTORS AT THE END
     EVAL[ND] := EVAL[I];
     LOOP(N) E[ND,N] := E[I,N];
     ND := ND - 1;
     NEG := NEG - 1;
     I := I - 1;
END;
I := 1;
NZRO := 0;
NEG := 0;
ND := M;
WHILE I LEQ KVALLEY AND I LEQ ND DO
BEGIN     % DO POWER ITERS AND SET UP POSITIVE
          % EIGEN VALUES
     N := GETEIGENVECTOR(I);
     EVALN := EVAL[N];
     LOOP(J) EWORK[J] := E[N,J];
     POWER(EVALN,EWORK,100);
     IF EVALN = 0.0 THEN
     BEGIN     % ZERO EIGEN VALUE
          L := ND;
          ND := I;
          J := I;
          WHILE M LEQ L := L + 1 DO
          BEGIN     % SHIFT NEGATIVE EIGEN VECTORS UP
               J := J + 1;
               DELS[J] := DELS[L];
               EVAL[J] := EVAL[L];
               LOOP(N) E[J,N] := E[L,N];
          END;
          WHILE M LEQ J := J + 1 DO
          BEGIN     % ZERO OUT REMAINING VALUES
               NZRO := NZRO + 1;
               DELS[J] := 0.0;
               EVAL[J] := 0.0;
               LOOP(N) E[J,N] := 0.0;
          END;
          N := 0;
          I := ND + 1;
     END ELSE
     IF EVALN > 0 THEN
     BEGIN     % POSITIVE EIGEN VALUE
          N := I;
          I := I + 1;
          DELSI := 0.0;
```

```
            LOOP(J) DELSI := DELSI + EWORK[J] * GRAD[J];
            DELS[N] := - DELSI / ABS(EVALN);
         END ELSE
         BEGIN     % NEGATIVE EIGEN VALUE IS STORED
            NEG := NEG + 1;
            N := ND;
            ND := ND - 1;
         END;
         EVAL[N] := EVALN;
         LOOP(J) E[N,J] := EWORK[J];
         IF N NEQ KVALLEY AND N NEQ I THEN
            DEFLATE(EVALN,EWORK);
      END;
      FOR I := ND + 1 STEP 1 UNTIL KVALLEY DO
      IF EVAL[I] NEQ 0.0 THEN
      BEGIN     % PICK UP NEGATIVE EIGEN VALUES
         DELSI := 0.0;
         LOOP(J) DELSI := DELSI + E[I,J] * GRAD[J];
         DELS[I] := - DELSI / ABS(EVAL[I]);
      END;
      DLSMX := 0.0;
      FOR I := 1 STEP 1 UNTIL K DO
         IF ABS(DELS[I]) > DLSMX THEN DLSMX := ABS(DELS[I]);
END OF PROCEDURE UPDATEGRADHESSIANEIGENVECTORS;


      SETUP;


      WHILE NOT CONVERGED DO
      BEGIN
         IF ACCELERATIONPOSSIBLE THEN     % GET DIR
         DO BEGIN     % ACCELERATE ALONG THE VALLEY
            UPDATEDS;     % UPDATE THE STEP SIZE, DS
            LOOP(I) XWORK[I] := X[I] + DS * DIR[I];
            FUNC(XWORK,F);
            STOREF0;
            NFL := NFL + 1;
         END UNTIL FINCREASE;     % STEP UNTIL INCREASE IN F
         WHILE NOTONVALLEY DO
         BEGIN     % GET BACK ON THE VALLEY
            GETDIR;     % NEWTON'S DIRECTION
            IF KVALLEY > K THEN INCLUDEKVALLEY;
            LINESEARCH(XWORK,F);
            STOREF0;
            IF NEWXWORK THEN
               UPDATEGRADHESSIANEIGENVECTORS(XWORK);
         END;
```

```
          IF MINONHYPERVALLEY THEN HYPERCROSSECTION;
     END OF THE ALGORITHM;


END OF PROCEDURE BANANA;



     NFUNC := 0;
     NGRAD := 0;
     CPUT := TIME(12);
     BANANA(X0,F0,M);



EOP:
     CPUT := (TIME(12) - CPUT) / 2.4@5;
 WRITE(LINE,*//,NFUNC,NGRAD,NFUNC + M * NGRAD,NFLINE,NLINE,
    CPUT,NHV,NDEF,WNTN,X0,F0);

END.
```

APPENDIX D

Detailed Results for the Standard Test Problems

This appendix gives detailed results in tabular form for the 19 standard test problems. The number of function calls has been calculated assuming that a one sided finite difference method was used to obtain the gradient of the objective function. This adjustment was not done for quasi-Newton as it takes its own numerical derivatives. Therefore, the computer times listed for quasi-Newton are misleading because the other algorithms used analytical derivatives which take less computer time. However, all the computer times have been included for completeness. These computer times are actual CPU seconds, not wall clock time. However, these times will vary slightly depending on what else is in the computer. The number of function calls used in the line searches is also included because BANANA does not always perform line searches. Newton steps have been included in this number. The IMSL conjugate gradient algorithm does not perform line searches. Therefore, its number of calls per line search is always 1.00. When the subvariety has been expanded to include the entire solution space,

ARTHUR LAKES LIBRARY
COLORADO SCHOOL of MINES
GOLDEN, COLORADO 80401

and the minimum on the current hypervalley is bracketed, BANANA performs Newton searches to find the minimum of the objective function. This number is also included in the tables.

Table D-1. Detailed results for Function 1.

	BANANA	modified Newton	Conjugate Gradient	Quasi-Newton
Adjusted function evaluations	2177	*	59801	32077
Function at min	$2.6*10^{-15}$		$8.8*10^{-14}$	$4.0*10^{-11}$
CPU time (sec)	8.405		7.260	11.409
Function calls	245		4601	2905
Gradient calls	161		4600	
Function calls in line search	111			2905
Line searches	89			**
Calls per search	1.25		1.00	
Newton searches	0			

* Newton's direction did not give a decrease in the objective function.
** Quit due to rounding errors.

Table D-2. Detailed results for Function 2.

	BANANA	modified Newton	Conjugate Gradient	Quasi-Newton
Adjusted function evaluations	407	667	2472	1239
Function at min	$8.0*10^{-18}$	$3.9*10^{-15}$	$1.9*10^{-14}$	$7.5*10^{-14}$
CPU time (sec)	0.580	0.201	0.269	0.351
Function calls	83	229	354	273
Gradient calls	54	73	353	
Function calls in line search	45	229		273
Line searches	29	73		117
Calls per search	1.55	3.14	1.00	2.33
Newton searches	1	73		117

Table D-3. Detailed results for Function 3.

	BANANA	modified Newton	Conjugate Gradient	Quasi-Newton
Adjusted function evaluations	222	354	1201	594
Function at min	$9.1*10^{-21}$	$1.1*10^{-15}$	$2.8*10^{-14}$	$5.2*10^{-15}$
CPU time (sec)	0.211	0.075	0.099	0.135
Function calls	62	166	241	210
Gradient calls	40	47	240	
Function calls in line search	31	166		210
Line searches	22	47		85
Calls per search	1.41	3.53	1.00	2.47
Newton searches	1	47		85

Table D-4. Detailed results for Function 4.

	BANANA	modified Newton	Conjugate Gradient	Quasi-Newton
Adjusted function evaluations	72	92	223	209
Function at min	$2.2*10^{-20}$	$1.2*10^{-18}$	$5.1*10^{-19}$	$1.5*10^{-17}$
CPU time (sec)	0.042	0.019	0.019	0.038
Function calls	40	62	75	113
Gradient calls	16	15	74	
Function calls in line search	19	62		113
Line searches	12	15		43
Calls per search	1.58	4.13	1.00	2.63
Newton searches	5	15		43

Table D-5. Detailed results for Function 5.

	BANANA	modified Newton	Conjugate Gradient	Quasi-Newton
Adjusted function evaluations	33	127	220	116
Function at min	$8.6*10^{-18}$	$2.1*10^{-15}$	$1.7*10^{-15}$	$5.1*10^{-18}$
CPU time (sec)	0.032	0.024	0.020	0.021
Function calls	19	91	74	96
Gradient calls	7	18	73	
Function calls in line search	10	91		96
Line searches	6	18		20
Calls per search	1.67	5.06	1.00	4.80
Newton searches	4	18		20

Table D-6. Detailed results for Function 6.

	BANANA	modified Newton	Conjugate Gradient	Quasi-Newton
Adjusted function evaluations	169	22400	824	382
Function at min	$8.3*10^{-16}$	$9.98*10^{-14}$	$6.2*10^{-14}$	$2.1*10^{-15}$
CPU time (sec)	0.118	4.482	0.068	0.079
Function calls	64	5618	207	151
Gradient calls	35	5594	206	
Function calls in line search	29	5618		151
Line searches	20	5594		64
Calls per search	1.45	1.00	1.00	2.36
Newton searches	1	5594		64

Table D-7. Detailed results for Function 7.

	BANANA	modified Newton	Conjugate Gradient	Quasi-Newton
Adjusted function evaluations	235	139	566	265
Function at min	$2.8*10^{-14}$	$4.0*10^{-14}$	$9.2*10^{-14}$	$5.7*10^{-14}$
CPU time (sec)	0.179	0.038	0.048	0.059
Function calls	83	55	114	105
Gradient calls	38	21	113	
Function calls in line search	42	55		105
Line searches	30	21		34
Calls per search	1.40	2.62	1.00	3.09
Newton searches	13	21		34

Table D-8. Detailed results for Function 8.

	BANANA	modified Newton	Conjugate Gradient	Quasi-Newton
Adjusted function evaluations	228	141	1551	213
Function at min	$2.1*10^{-14}$	$2.7*10^{-14}$	$9.3*10^{-14}$	$9.9*10^{-14}$
CPU time (sec)	0.191	0.037	0.130	0.049
Function calls	72	53	311	85
Gradient calls	39	22	310	
Function calls in line search	42	55		85
Line searches	31	21		29
Calls per search	1.35	2.62	1.00	2.93
Newton searches	13	21		29

Table D-9. Detailed results for Function 9.

	BANANA	modified Newton	Conjugate Gradient	Quasi-Newton
Adjusted function evaluations	51	133	196	110
Function at min	$1.2*10^{-20}$	$3.3*10^{-24}$	$9.8*10^{-18}$	$1.7*10^{-14}$
CPU time (sec)	0.041	0.024	0.025	0.024
Function calls	35	89	66	64
Gradient calls	8	22	65	
Function calls in line search	10	89		64
Line searches	5	22		19
Calls per search	2.00	4.05	1.00	3.37
Newton searches	2	22		19

Table D-10. Detailed results for Function 10.

	BANANA	modified Newton	Conjugate Gradient	Quasi-Newton
Adjusted function evaluations	106	88	196	139
Function at min	$4.3*10^{-18}$	$1.3*10^{-23}$	$7.5*10^{-17}$	$3.4*10^{-15}$
CPU time (sec)	0.061	0.025	0.020	0.028
Function calls	58	52	66	79
Gradient calls	24	18	65	
Function calls in line search	28	52		79
Line searches	17	18		27
Calls per search	1.65	2.89	1.00	2.93
Newton searches	7	18		27

Table D-11. Detailed results for Function 11.

	BANANA	modified Newton	Conjugate Gradient	Quasi-Newton
Adjusted function evaluations	55	*	43	69
Function at min	$1.4*10^{-20}$		$3.7*10^{-14}$	$1.4*10^{-20}$
CPU time (sec)	0.035		0.006	0.016
Function calls	35		15	39
Gradient calls	10		14	
Function calls in line search	16			39
Line searches	7			13
Calls per search	2.29		1.00	3.00
Newton searches	2			13

* Newton's direction did not give a decrease in the objective function.

Table D-12. Detailed results for Function 12.

	BANANA	modified Newton	Conjugate Gradient	Quasi-Newton
Adjusted function evaluations	50	49	149	116
Function at min	$3.1*10^{-20}$	0.0	$3.4*10^{-14}$	$8.7*10^{-14}$
CPU time (sec)	0.045	0.011	0.013	0.026
Function calls	20	28	38	53
Gradient calls	10	7	37	
Function calls in line search	12	28		53
Line searches	5	7		19
Calls per search	2.40	4.00	1.00	2.79
Newton searches	3	7		19

Table D-13. Detailed results for Function 13.

	BANANA	modified Newton	Conjugate Gradient	Quasi-Newton
Adjusted function evaluations	40	*	160	111
Function at min	$2.5*10^{-20}$		$1.2*10^{-14}$	$2.5*10^{-21}$
CPU time (sec)	0.043		0.023	0.024
Function calls	18		54	65
Gradient calls	11		53	
Function calls in line search	11			65
Line searches	9			21
Calls per search	1.22		1.00	3.10
Newton searches	4			21

* Newton's direction did not give a decrease in the objective function.

Table D-14. Detailed results for Function 14.

	BANANA	modified Newton	Conjugate Gradient	Quasi-Newton
Adjusted function evaluations	46	*	70	79
Function at min	$6.3*10^{-19}$		$3.6*10^{-17}$	$1.2*10^{-17}$
CPU time (sec)	0.035		0.011	0.019
Function calls	28		24	41
Gradient calls	9		23	
Function calls in line search	15			41
Line searches	6			17
Calls per search	2.50		1.00	2.41
Newton searches	2			17

* Newton's direction did not give a decrease in the objective function.

Table D-15. Detailed results for Function 15.

	BANANA	modified Newton	Conjugate Gradient	Quasi-Newton
Adjusted function evaluations	39	32	67	68
Function at min	$1.1*10^{-18}$	0.0	$6.2*10^{-18}$	$7.2*10^{-22}$
CPU time (sec)	0.038	0.008	0.007	0.016
Function calls	23	22	23	34
Gradient calls	8	5	22	
Function calls in line search	12	22		34
Line searches	6	5		15
Calls per search	2.00	4.40	1.00	2.27
Newton searches	3	5		15

Table D-16. Detailed results for Function 16.

	BANANA	modified Newton	Conjugate Gradient	Quasi-Newton
Adjusted function evaluations	39	34	55	56
Function at min	$2.9*10^{-20}$	0.0	$1.5*10^{-18}$	$2.4*10^{-18}$
CPU time (sec)	0.040	0.010	0.006	0.017
Function calls	21	22	19	32
Gradient calls	9	6	18	
Function calls in line search	12	22		32
Line searches	7	6		11
Calls per search	1.71	3.67	1.00	2.91
Newton searches	3	6		11

Table D-17. Detailed results for Function 17.

	BANANA	modified Newton	Conjugate Gradient	Quasi-Newton
Adjusted function evaluations	19	23	37	52
Function at min	$2.2*10^{-19}$	$2.5*10^{-20}$	$3.0*10^{-19}$	$1.1*10^{-18}$
CPU time (sec)	0.036	0.007	0.005	0.012
Function calls	9	15	13	32
Gradient calls	5	4	12	
Function calls in line search	7	15		32
Line searches	3	4		9
Calls per search	2.33	3.75	1.00	3.56
Newton searches	1	4		9

Table D-18. Detailed results for Function 18.

	BANANA	modified Newton	Conjugate Gradient	Quasi-Newton
Adjusted function evaluations	17	8	13	47
Function at min	$1.1*10^{-21}$	$1.1*10^{-21}$	$9.6*10^{-18}$	0.0
CPU time (sec)	0.030	0.004	0.003	0.012
Function calls	11	6	5	29
Gradient calls	3	1	4	
Function calls in line search	8	6		29
Line searches	1	1		7
Calls per search	8.00	6.00	1.00	4.14
Newton searches	0	1		7

Table D-19. Detailed results for Function 19.

	BANANA	modified Newton	Conjugate Gradient	Quasi-Newton
Adjusted function evaluations	30	9	31	71
Function at min	$8.5*10^{-22}$	$2.5*10^{-20}$	$1.8*10^{-19}$	$3.9*10^{-34}$
CPU time (sec)	0.020	0.007	0.005	0.016
Function calls	15	6	11	41
Gradient calls	5	1	10	
Function calls in line search	11	6		41
Line searches	1	1		8
Calls per search	11.00	6.00	1.00	5.13
Newton searches	0	1		8

APPENDIX E

Data Used in the Oil Well Test Problems

History data for the first three oil well test problems were generated on the computer using the input data given in Tables E-1 to E-3. Table E-4 gives the layer properties from the minimum sum of squares objective function. Permeability and skin factors do not agree as close as initial pressure due to the high degree of correlation of permeability and skin as indicated by Equation (24). History data consisted of layer rate data taken at the end of each of the three eight hour production intervals, seven well pressure points taken during the last producing period and 28 well pressure points taken during the shut in period. The first two test problems were taken directly from this data with the two different starting points given in Table E-5.

Table E-1. Data used to generate the well test data.

Number of grid blocks in the radial direction	25
Number of layers	5
Edge of the reservoir	3000 ft
Well radius	0.354 ft
System compressability	0.000021/psi
Formation volume factor	1.8 RB/STB
Reservoir depth	12000 ft
Oil density	45.0 lbm/cu ft
Oil viscosity	0.3 cp
Porosity	0.125
Well bore storage constant	0.05 RB/psi

Table E-2. Rate schedule for the simulation used to generate the well test data.

Flow Rate (STB/D)	Incremental Time (hr)	Cumulative Time (hr)
25000.0	3840.0	3840.0
20000.0	8.0	3848.0
15000.0	8.0	3856.0
5000.0	8.0	3864.0
24000.0	40.0	3904.0
0.0	80.0	3984.0

Table E-3. Layer properties used to generate the well test data.

	layer				
	1	2	3	4	5
Skin	5.0	3.0	7.0	2.0	4.0
Radial permeability	25.0	17.0	63.0	47.0	52.0
Vertical permeability	0.0	0.0	0.0	0.0	0.0
Layer thickness	110.0	30.0	200.0	58.0	75.0
Initial layer pressure	5743.0	5825.0	5930.0	6010.0	6250.0

Table E-4. Layer properties obtained from the minimum sum of squares objective function.

	layer				
	1	2	3	4	5
Skin	9.20	9.55	5.04	3.16	3.79
Radial permeability	33.75	28.50	54.25	53.00	51.00
Initial layer pressure	5680.0	5720.0	5960.0	6000.0	6240.0

Table E-5. Starting points for the first two well test problems.

	Problem 1	Problem 2
Radial permeability	50.0	25.0
Skin	1.0	10.0
Initial pressure	7000.0	4000.0

The third test problem is the same as the second except that random errors were added to the simulated data to make it more like real data. For actual well tests, the pressure data are very accurate. Therefore, these data were not modified. However, there are errors in the layer rate data and in the well production rates. Therefore, the layer rate data for the first four layers were modified by the normally distributed (mean = 0, variance = 25) random percentages given in Table E-6. The rate for the fifth layer was adjusted so that the layer rates sum to the well rate. The rate for the first production period was raised 20.546 percent and the rate for the last production period was

raised 5.004 percent. All these random percentages were generated by the same program. This program averaged twelve uniform (0,1) random variables obtained from the IMSL subroutine GGUBS to obtain a normal (mean = 0, variance = 1) random variable. Each of these random variables was then converted to the desired random percentages. Finally, the fourth test problem was taken from real data where 12 variables were adjusted to obtain a match. These data are priority data and thus cannot be released at this time.

Table E-6. Random percentages used to put random errors into the simulated well test data.

		problem			
PLT	Rate Period	1	2	3	4
	1	11.012	10.483	9.735	2.191
	2	-3.541	-10.093	-7.830	6.828
	3	5.320	10.432	7.581	6.071