

DOE/BC/14126-16
(DE90000249)

**A STUDY ON NEWTON RELATED NONLINEAR METHODS
IN WELL TEST ANALYSIS, PRODUCTION SCHEDULE
OPTIMIZATION AND RESERVOIR SIMULATION**

SUPRI TR 70

**By
Jawahar Barua**

August 1990

Performed Under Contract No. FG19-87BC14126

**Stanford University
Stanford, California**



**Bartlesville Project Office
U. S. DEPARTMENT OF ENERGY
Bartlesville, Oklahoma**

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

Available to DOE and DOE contractors from the Office of Scientific and Technical Information, P.O. Box 62, Oak Ridge, Tennessee 37831; prices available from (615)576-8401, FTS 626-8401.

Available to the public from the National Technical Information Service, U.S. Department of Commerce, 5285 Port Royal Rd., Springfield, Virginia 22161.

Price: Printed Copy A06
Microfiche A01

A STUDY ON NEWTON RELATED NONLINEAR METHODS
IN WELL TEST ANALYSIS, PRODUCTION SCHEDULE
OPTIMIZATION AND RESERVOIR SIMULATION

SUPRI TR 70

By
Jawahar Barua

August 1990

Work Performed Under Contract No. FG19-87BC14126

Prepared for
U.S. Department of Energy
Assistant Secretary for Fossil Energy

Thomas B. Reid, Project Manager
Bartlesville Project Office
P.O. Box 1398
Bartlesville, OK 74005

Prepared by
Stanford University
Petroleum Research Institute
Stanford, CA 94305-4042

TABLE OF CONTENTS

	<u>Page</u>
TABLE OF CONTENTS.....	iii
LIST OF TABLES.....	v
LIST OF FIGURES.....	vi
ACKNOWLEDGEMENTS.....	vii
ABSTRACT.....	viii
1. INTRODUCTION.....	1
1.1 Solving Nonlinear Equations.....	1
1.1.1 Reservoir Simulation.....	2
1.1.2 Linear Formulations.....	2
1.1.3 Nonlinear Formulations.....	3
1.2 Minimizing or Maximizing a Function.....	3
1.2.1 Automated Well Test Analysis.....	4
1.3 Optimization.....	4
1.3.1 Parallel Reservoir Simulation.....	5
2. COMPARISON OF FIRST AND SECOND ORDER METHODS IN AUTOMATED WELL TEST ANALYSIS.....	6
2.1 Newton's Method.....	6
2.2 The Nonlinear Least-Squares Problem.....	8
2.3 Description of the Procedures.....	9
2.3.1 The Method of Steepest Descent.....	9
2.3.2 The Gauss Method.....	10
2.3.3 The Marquardt Method.....	10
2.3.4 The Newton-Greenstadt Method.....	11
2.3.5 Penalty Functions.....	12
2.3.6 Line Search.....	12
2.3.7 Solving the Matrix Problem.....	13
2.4 Evaluation of the Algorithms.....	14
2.4.1 Storage and Skin, Case 1.....	14
2.4.2 Storage and Skin, Case 2.....	21
2.4.3 Storage and Skin, Case 3.....	23
2.4.4 Storage and Skin, Case 4.....	27
2.4.5 Double-Porosity Case.....	27
2.4.6 Storage and Skin, Case 5.....	30
2.5 Summary.....	30
3. PRODUCTION AND INJECTION SCHEDULE OPTIMIZATION.....	34
3.1 Description of the Problem.....	34
3.2 Mathematical Model.....	34
3.3 Optimization.....	36
3.3.1 Newton's Method with Finite Differences.....	36
3.3.2 Quasi-Newton Methods for Optimization.....	38
3.4 Optimization Experiments with the Model.....	38
3.4.1 Optimizing a Linear Combination of Objectives.....	39
3.4.2 Effect of Steam Properties.....	42
3.4.3 Effect of Cycle Parameters.....	42
3.5 Summary.....	46

4. QUASI-NEWTON METHODS IN RESERVOIR SIMULATION.....	47
4.1 Previous Work.....	47
4.2 Reservoir Simulation.....	47
4.3 Quasi-Newton Methods for Systems of Equations.....	50
4.4 Broyden's Update.....	50
4.4.1 Properties of Broyden's Update.....	51
4.4.2 Implementing the Update Efficiently.....	52
4.4.3 Restarting Updates.....	54
4.4.4 Summary of the Method.....	55
4.5 Two-Phase Experiments with the QN Method.....	55
5. PARALLEL RESERVOIR SIMULATION.....	58
5.1 Parallel Computing.....	58
5.2 Amdahl's Law.....	58
5.3 Parallel Matrix Solution.....	59
5.4 Objectives of Current Study.....	60
5.5 Shared-Memory and Message-Passing Architectures.....	60
5.6 Programming a Shared-Memory Machine.....	62
5.6.1 The CREATE Primitive.....	63
5.6.2 The LOCK Primitive.....	63
5.6.3 The BARRIER Primitive.....	64
5.7 Building a Parallel Reservoir Simulator.....	65
5.7.1 Parallel Construction of f and J	65
5.7.2 Parallelizing Vector Dot Products.....	67
5.7.3 Miscellaneous Linear Algebra Routines.....	68
5.7.4 Globally Shared Variables.....	68
5.8 Parallel Experiments with the Quasi-Newton Method.....	69
5.9 Comparing Newton and Quasi-Newton Methods.....	72
5.9.1 Variable Substitution.....	72
5.9.2 Test Conditions.....	72
5.9.3 First Order Techniques.....	73
5.9.4 Accelerated Techniques.....	74
5.9.5 Effect of Number of Updates.....	78
5.9.6 Effect of Simulation Parameters.....	80
5.9.7 Performance on Approximate Jacobians.....	80
5.10 Solving the Matrix Equations in Parallel.....	83
5.10.1 Adding Constraints.....	85
5.10.2 Solving Matrix and Constraint Equations in Parallel.....	87
5.11 Experiments with the Parallel Simulator.....	87
5.11.1 Comparison of Parallel Algorithms.....	91
5.11.2 Effect of Constraints on Speedup and Total Time.....	94
5.11.3 Effect of Input/Output on Speedup.....	94
5.11.4 Speedup of Various Tasks in Simulation.....	94
6. CONCLUSIONS.....	103
NOMENCLATURE.....	105
REFERENCES.....	107

LIST OF TABLES

	<u>Page</u>
2.1 Data for storage and skin tests	15
2.2 Initial guesses, bounds and best-fit values for storage and skin test are listed in Table 2.4	16
2.3 Data for double porosity tests	17
2.4 Initial guesses, bounds and best-fit values for double porosity test	18
3.1 Optimizing net production (4 parameters).....	40
3.2 Optimizing a linear combination of objectives	41
3.3 Optimizing with higher steam temperature.....	43
3.4 Optimizing with higher steam quality.....	44
3.5 Optimizing with balanced steam properties	45
5.1 Comparison of direct and first order iterative methods	75
5.2 Comparison of iterative methods	77
5.3 Comparison of number of updates.....	79
5.4 Depletion gas drive	81
5.5 Doubled oil and gas rates	81
5.6 Performance on approximate Jacobians	82
5.7 Effect of residual constraints on linear convergence	88
5.8 Effect of Z-line length on residual constraints	89
5.9 Comparison of some parallel algorithms	92

LIST OF FIGURES

	<u>Page</u>
2.1 Storage and skin test	19
2.2 Convergence, Case 1	19
2.3 Eigenvalues of Newton's method, Case 1	20
2.4 Eigenvalues of Newton-Greenstadt method, Case 1	20
2.5 Eigenvalues of Gauss-Marquardt method, Case 1	22
2.6 Convergence, Case 2	22
2.7 Eigenvalues of Gauss-Marquardt method, Case 2	24
2.8 Convergence, Case 3	24
2.9 Eigenvalues of Newton's method, Case 3	25
2.10 Eigenvalues of Newton-Greenstadt method, Case 3	25
2.11 Step lengths of Newton-Greenstadt (a) method, Case 3	26
2.12 Convergence, Case 4	26
2.13 Gauss-Greenstadt, Case 4	28
2.14 Double-porosity test case	28
2.15 Double-porosity convergence	29
2.16 Eigenvalues for Gauss-Marquardt	29
2.17 Convergence, double-porosity test, 6 parameters	31
2.18 Double-porosity test eigenvalues, 6 parameters	31
2.19 Fit with double porosity model	32
2.20 Eigenvalues for Gauss-Marquardt method	32
4.1 Newton's method for solving nonlinear systems of equations	49
4.2 Total time on 2-D oil-water system	57
4.3 Time on 2-D and 3-D oil-water systems	57
5.1 Shared-memory and message-passing architectures	61
5.2 Use of BARRIERS on 2 processors	64
5.3 Program flow chart on 2 processors	66
5.4 Parallel construction of f and J	67
5.5 Time on serial vs time on parallel	70
5.6 Speedup vs problem size	70
5.7 Speedup on 4 processors	71
5.8 Speedup on 12 processors	71
5.9 Red-black ordering of 5×4 grid	84
5.10 The Orthomin algorithm	84
5.11 Constrained residual preconditioning for Orthomin	90
5.12 Effect of problem size	93
5.13 Speedup on 12 processors	93
5.14 Speedup on 14 processors	95
5.15 Effect of corrections on speedup on $10 \times 10 \times 3$ problem	95
5.16 Effect of corrections on speedup on $6 \times 6 \times 3$ problem	96
5.17 Effect of corrections on total time on 1 processor	96
5.18 Effect of corrections on total time on 14 processors	97
5.19 Effect of I/O on parallel speedup	97
5.20 Time spent on the major tasks in simulating the test running in parallel	99
5.21 Speedup of solve step for 1 day time step	99
5.22 Speedup of solve step for 100 day time step	100
5.23 Speedup of forming and factorizing Jacobian	100
5.24 Example code for factorizing by lines	101

ACKNOWLEDGEMENTS

Financial support was provided by the Stanford University Petroleum Research Institute (SUPRI) under DOE contract DE-F619-87BC14126 and by the SUPRI Industrial Advisory Committee.

ABSTRACT

This is a study on the use of alternative nonlinear methods in automated well test analysis, production and injection schedule optimization and in reservoir simulation.

In automated well test analysis the advantages and disadvantages of second-order partial derivatives are investigated. Newton's method is shown to be prone to difficulties, however by adjusting the eigenvalues of the Hessian matrix, the performance can be substantially improved.

In optimizing the cyclic steam injection process, Newton's method is compared with the Quasi-Newton method using a simplified model to simulate the process. The Quasi-Newton method does significantly better than Newton's method in saving function evaluations. Specific operating strategies for the process are identified: the need to eliminate soak, the need for greatly increased steam volumes and temperatures, and the need to optimize a combination of economic objectives.

The two methods are then compared in reservoir simulation. Tests show that while it is possible to use the Quasi-Newton method to build up inverse Jacobians as the iterations proceed, for difficult problems the method requires the use of matrix solution techniques. The method then becomes directly comparable to Newton's method. Tests show that depending upon the linear scheme used, and the difficulty of the problem, the Quasi-Newton method may prove to be less expensive than Newton's method in certain cases.

The study also addresses the issue of building scalable parallel reservoir simulators. Residual constraints are used to improve the robustness of the parallel matrix solution scheme. The solution of the constraint matrix is shown to be a critical point in achieving good performance on a parallel machine.

1. INTRODUCTION

Many computational tasks in petroleum engineering require the solution of nonlinear equations or the optimization of a nonlinear function. Both problems are very similar. To find an extremum of a function we look for the point where its gradient, \mathbf{g} , becomes zero, i.e. we solve $\mathbf{g}(\mathbf{x}) = 0$. This is identical to the problem of solving a nonlinear system of equations $\mathbf{f}(\mathbf{x}) = 0$. Because of this similarity, the same nonlinear iterative techniques apply to both problems, the classical technique being Newton's method.

Although both problems are similar, in practical applications the performance of Newton's method differs significantly from one application to another. The modifications made to Newton's method reflect the issues that are important for each problem.

When solving nonlinear equations, Newton's method usually converges rapidly. So modifications to the method concentrate on reducing the costs involved.

When minimizing a function, the convergence of Newton's method becomes an issue; this has two aspects to it. Firstly, convergence to the solution of $\mathbf{g}(\mathbf{x}) = 0$ often requires some modification to the method. Secondly, even if the method solves $\mathbf{g}(\mathbf{x}) = 0$, this may not be the solution of the original minimization problem. Modifications to the Newton's method for this case therefore attempt to improve convergence. Cost is a problem only if the derivatives are time consuming to evaluate, for example if they have to be evaluated by finite-difference.

So zero-finding and function minimization are at the same time very similar and also very different. This study explores some alternatives to Newton's method in automated well test analysis, optimization and reservoir simulation.

1.1. SOLVING NONLINEAR EQUATIONS

The solution of the problem

$$\mathbf{f}(\mathbf{x}) = 0$$

is required in many forms of engineering problems. For example, such a solution is required at each time step in fully implicit reservoir simulation. If \mathbf{f} is linear in the unknowns (or can be linearized in some way) then the system can be written as $\mathbf{A}\mathbf{x} = \mathbf{b}$, where \mathbf{A} is the matrix of the coefficients of \mathbf{x} and \mathbf{b} the vector of constant terms from each equation. The solution is then obtained in one step.

Iterative techniques are needed when the problem is nonlinear. Newton's method for solving this system is based on starting with an estimate for \mathbf{x} and constructing a locally linear model of the function, using the truncated Taylor series expansion

$$\mathbf{f}(\mathbf{x}_k + \Delta\mathbf{x}_k) \approx \mathbf{f}(\mathbf{x}_k) + \mathbf{J}(\mathbf{x}_k)\Delta\mathbf{x}_k$$

where \mathbf{J} is the Jacobian matrix, with elements $J_{ij} = (\partial f_i)/(\partial x_j)$. We wish to take a step to the zero of the linear model, so we set $\mathbf{f}(\mathbf{x}_k + \Delta\mathbf{x}) = 0$ and obtain Newton's method for a system of equations (also termed the Newton-Raphson method)

$$\mathbf{J}_k\Delta\mathbf{x}_k = -\mathbf{f}_k$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta\mathbf{x}_k$$

Newton's method is quadratically convergent when close to the solution, so it is a very attractive method. However the matrix solution required at each iteration can be expensive when the number of unknowns is large.

1.1.1. Reservoir Simulation

In fully implicit reservoir simulation all fluid and reservoir properties are evaluated at the unknown level of time, and the resulting system of equations $f(x) = 0$ is solved by Newton's method. A good initial guess for x is always available since the value at the previous time step can be used. Moreover by reducing the time step size the initial guess can be made as close as desired to the solution. Convergence of Newton's method in simulation is therefore always possible, and will usually approach a quadratic rate of convergence. However, there are a very large number of unknowns (the number of grid blocks times the number of equations at each grid block) to be found at each iteration, so a great deal of expense is involved.

To reduce this expense, several alternative methods to formulate the problem or to solve the nonlinear system are possible.

1.1.2. Linear Formulations

The most economical techniques for reservoir simulation avoid Newton's method altogether and formulate the simulation as a linear problem. This is done by evaluating fluid and reservoir properties at the old (known) level of time or by assuming that the properties change linearly with respect to x . With linear formulations, the solution is obtained in one step so substantial savings result. Depending on the details of the formulation, further savings are possible, for example:

- The IMPES method treats pressures as the only implicit unknowns, so the matrix problem has only one equation per grid block. Fluid and reservoir properties are evaluated at the old level of time. The small size of the matrix and the lack of iteration together make IMPES the fastest method for the problem classes for which it is suitable.
- The Sequential Solution method requires the sequential solution of an IMPES type matrix for each equation at a grid block. If, for example, there are three equations to be solved at each block (pressure, oil saturation and water saturation), the cost of solving three n th order matrices is less than the cost of solving one matrix of order $3n$, so this method also saves on the matrix.
- The Linearized Implicit method is the most expensive. Fluid and rock properties are assumed to vary linearly with x , so the method is equivalent to the first iteration of Newton's method. However by avoiding iterations it is still at least three to four times cheaper than Newton's method.

While all linearized methods offer savings, in return they sacrifice stability with respect to time step size; all are less stable than the fully implicit Newton's method (Aziz and Settari, 1979).

1.1.3. Nonlinear Formulations

The more stable methods treat simulation as a nonlinear problem and require iteration. To cut down on expense, the approach often used is to save time on the matrix solution step by some means and make up for this by performing a few extra iterations.

- The Adaptive Implicit method (Thomas and Thurnau, 1981) treats a subset of the reservoir unknowns implicitly. The remaining unknowns are solved explicitly and eliminated first, so the size of the matrix problem is reduced to just that needed to handle the implicit variables. This method has become quite popular since it retains the strengths of Newton's method (i.e. implicitness) where needed, while saving on the expense.
- The Inexact Adaptive Newton method (Bertiger and Kelsey, 1985) is a variant of the adaptive implicit method (it uses a different treatment for the diagonal terms). Bertiger and Kelsey report both methods achieving 5 to 30 time savings compared to the fully implicit method.
- The Inexact Newton method (Dembo *et al.*, 1982) uses the fully implicit equations unchanged. The accuracy of the solution is varied instead - starting with a low accuracy and solving to increasingly higher accuracy as the iterations proceed. Time is saved by solving to low accuracy at the earlier iterations.
- The Quasi-Newton method (see Dennis and Schnabel, 1983) is a different approach to solving the nonlinear problem $f(x) = 0$. The method stores the values of f from each iteration and can use them to build up an approximate Jacobian. In practical application it reduces the number of Jacobian evaluations and factorizations needed to reach the solution, at the expense of convergence rate. The Quasi-Newton method has had very good success in optimization and some success in solving nonlinear equations in other fields. For reservoir simulation Nghiem (1983) proposed a method dubbed the QNSS (Quasi-Newton Successive Substitution) and reported success with it in solving the pressure equation in compositional simulation.
- The latter part of the study explores the application of the more popular Quasi-Newton methods to reservoir simulation problems. It shows that the Quasi-Newton method can offer time savings in certain cases but suffers from other drawbacks. If the Jacobian matrix cannot be formed accurately, or if the number of unknowns at each grid block is large then the method is a primary contender, otherwise it seems best suited to be a backup option for a primary method that is closer to the true Newton method.

1.2. MINIMIZING OR MAXIMIZING A FUNCTION

The problem of maximizing or minimizing a scalar valued function is closely related to that of nonlinear systems of equations, but there are some important differences. If the function, F , is linear in the unknowns, then it is unbounded on both sides and there is no minimum or maximum unless F is constant. A bounded minimum or maximum will only occur at constraints. Solutions to this type of problem are found using linear programming techniques.

Newton's method applies when the function is nonlinear. The nonlinear system is now based on finding the zero of the m gradient of F , i.e. we solve $g(x) = 0$. The resulting iterative scheme looks identical to Newton's method for systems of equations:

$$\mathbf{H}_k \Delta \mathbf{x}_k = -\mathbf{g}_k$$
$$\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta \mathbf{x}_k$$

except that the Jacobian matrix is replaced by the Hessian \mathbf{H} [$(H_{ij} = (\partial^2 F)/(\partial x_i \partial x_j))$] and the original vector valued function is replaced by the gradient \mathbf{g} ($g_i = \partial F/\partial x_i$).

For the kinds of optimization applications envisaged in petroleum engineering the number of unknowns is often much smaller than in reservoir simulation, so the cost of the matrix solution is no longer an issue. However a new set of problems appears:

- Since the gradient is zero at both minima and maxima, we cannot be sure that the stationary point reached will be the one desired.
- The Hessian may be ill-conditioned so numerical instability occurs during the matrix solution. This problem does not occur in reservoir simulation since the matrices are diagonally dominant and pivoting for numerical stability is usually not required (except among the phase equations at a particular grid block).
- Newton's method is based on a quadratic model of the function. At points far from the solution this model may be a poor fit so quadratic convergence may not be obtained. Unlike in simulation, we do not necessarily have a good starting guess to begin with.
- Evaluating the gradient and Hessian is expensive if they have to be found by finite-difference. This is the principal expense in optimization.

So unlike its application in simulation, a primary concern in Newton's method now is to ensure convergence to the required extremum. Cost savings now focus on minimizing the number of function evaluations needed to evaluate the gradient or Hessian, instead of on the matrix solution. So even though the basic method is unchanged, the important issues have changed quite a bit. We look at two different application areas for Newton's method and specific variations that attempt to solve such problems.

1.2.1. Automated Well Test Analysis

In this work, we studied the use of a variation of the Newton-Greenstadt (Greenstadt, 1967) method that solves both the problem of ill-conditioning and choice of direction. Tests show that this technique can greatly improve the convergence behavior of the otherwise fragile Newton method. Performance comes close to the popular Gauss-Marquardt method (Marquardt, 1963), but only exceeds it when initial guesses are good or when ill-defined parameters are present.

1.3. OPTIMIZATION

Following the study on automated well test analysis we looked at the problem of non-linear optimization of an EOR process. In automated well test analysis both gradient and Hessian can be obtained analytically. With less tractable functions they have to be obtained by finite difference, which can be very expensive. Quasi-Newton methods offer a way to save on this expense. Starting with the identity, \mathbf{I} , they build up an approximate Hessian using gradient

information picked up at each iteration. Such methods can potentially save on function evaluations and also retain positive-definiteness of the Hessian matrix to ensure a descent direction.

We compare Newton and Quasi-Newton methods on optimization of the cyclic steam injection process, and obtain some results pertinent to the process itself. Optimization shows that significant improvements in efficiency and productivity can be made by using optimal operating policies. It shows that operating at a high steam temperature makes the process much more amenable to large improvements in performance. Operating at a high steam quality is less effective. Also suggested by the optimizer are some changes that appear to be desirable under all circumstances - minimization of soak time and greatly increased steam injection volumes.

1.3.1. Parallel Reservoir Simulation

The optimization of cyclic steam injection was performed using a simplified mathematical model. In other applications a numerical simulator may be needed to more accurately model all phenomena. However since 50-100 full simulation runs may be needed to find an optimum, this may not be feasible on a conventional computer. The study finally looked at the problem of reservoir simulation on a parallel computer. We showed how a fully parallel simulator can be designed and constructed. We explored some parallel matrix solution schemes and show that parallel reservoir simulation may be viable for practical applications. Specifically we showed that the Red-Black ordering with Watts (1971) line corrections can provide a highly parallel solution scheme suitable for a very large number of processors, yet does not suffer greatly in serial against the best serial algorithm. We also investigated the effect of the line corrections on robustness and speedup and show that they represent a critical point in the parallelization of a simulator.

In summary, the purpose of this thesis can be stated as the exploration of nonlinear techniques in automated well test analysis, production schedule optimization and reservoir simulation. Parallel reservoir simulation became a related objective in the final study. The order of the material in the text follows the subjects of this study in the order stated above.

2. COMPARISON OF FIRST AND SECOND ORDER METHODS IN AUTOMATED WELL TEST ANALYSIS

Automated well test analysis has become a widely used technique in recent years. Several references to automated well test analysis have appeared in the past (Padmanabhan and Woo, 1976; Earlougher, 1977; Tsang et al., 1977; Padmanabhan, 1979; McEdwards, 1981). In the past one of the difficulties with automated well test analysis was the evaluation of the reservoir response functions in closed form. Rosa and Horne (1983) showed that it is possible to fit well test data directly to the Laplace space models of well transient behavior - by using the Stehfest (1970) algorithm to numerically invert pressure and its partial derivatives.

Comparisons of automated well test analysis with type-curve analysis or straight-line analysis techniques (Rosa and Horne, 1983; Barua and Horne, 1987; Guillot and Horne, 1984), showed some definite advantages. These include higher resolution than type-curve analysis, reduction of the danger of choosing incorrect straight lines, ease in handling of multiple flow rate history, reduced test duration etc.

Although automated well test analysis is often an improvement over conventional methods, practical problems sometimes occur in its application. One problem is when the data does not exactly match the model. In such cases there is not much one can do beyond using engineering judgement, data editing, and help from conventional techniques to obtain the best fit. Another problem occurs in the actual fitting process itself; ideally the program should converge rapidly to the optimum for the given data and the selected model. This does not always happen. Conventional analysis techniques can be used to give the method a good initial guess but it is also desirable for the nonlinear algorithm to be quickly convergent.

When ill-defined parameters are present, the fitting process tends to be adversely affected. Moreover, many common reservoir models are functions of different parameters over different time ranges (for example, the wellbore storage coefficient at early time, the permeability at intermediate time and the drainage volume at late time), and are often difficult problems to solve by nonlinear regression.

The most commonly used regression scheme is the Gauss-Newton method modified by Marquardt's algorithm - this is also termed the Levenberg-Marquardt method (Levenberg, 1944; Marquardt, 1963). This method usually converges rapidly so it has become popular for least-squares estimation. However the method sometimes runs into difficulties so there is a need for better estimation algorithms. In this study, a second order method based on Newton's method, specifically the Newton-Greenstadt method (Greenstadt, 1967), was examined to see if it could handle situations where the Gauss-Marquardt method had difficulty and also to see how it compared on more general problems.

2.1. NEWTON'S METHOD

The method which is central to all of those considered here is Newton's method for minimizing a nonlinear objective function. The approach used is to expand the objective function F in a Taylor series about the estimated value F_1 obtained at a particular solution x_1 and to truncate the series after the quadratic terms. This gives the Newton approximation F^* to the objective function:

$$F^* = F_1 + g_1^T (x - x_1) + \frac{1}{2} (x - x_1)^T H_1 (x - x_1) \quad (2.1)$$

where \mathbf{x} is the vector of n nonlinear parameters, \mathbf{g} the gradient vector:

$$\mathbf{g}_i = [\mathbf{g}_i] = \left[\frac{\partial F}{\partial x_j} \mid x_i \right] \quad (2.2)$$

and \mathbf{H} the Hessian matrix:

$$\mathbf{H}_i = [\mathbf{h}_{jk}] = \left[\frac{\partial^2 F}{\partial x_j \partial x_k} \mid x_i \right] \quad (2.3)$$

We wish to find the point \mathbf{x} where F^* is stationary so we equate to zero the gradient of F^*

$$\frac{\partial F^*}{\partial \mathbf{x}} = \mathbf{g}_i + \mathbf{H}_i (\mathbf{x} - \mathbf{x}_i) = 0 \quad (2.4)$$

which leads to the iterative scheme:

$$\mathbf{H}_i \mathbf{p}_i = -\mathbf{g}_i \quad (2.5)$$

and

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \rho \mathbf{p}_i \quad (2.6)$$

Where ρ is a step-length parameter obtained by a one-dimensional line search in the direction \mathbf{p}_i such that the next calculated value of F be as small as possible. Obtaining a precise value for ρ will be expensive so an approximate minimum is usually accepted. The line search procedure is an important component of an optimization program, as it can significantly improve convergence and also prevent divergence in some cases. In this work, the same procedure was used on all the test cases to separate the effect of line searches on the performance comparisons. In addition all the algorithms use penalty functions (Bard, 1974) to restrict the iterations to feasible regions.

If F is quadratic in the unknowns then Eq. 2.1 is exact and Newton's method converges in a single iteration. Usually the objective function may not be quadratic. But as the optimum is approached, the omitted terms in the Taylor series expansion become small, so Eq. 2.1 will closely approximate the function. Thus Newton's method will converge quadratically in the neighborhood of the optimum. At points remote from the optimum Newton's method approximates the function with the closest second order surface.

Newton's method chooses a potentially desirable direction when the contours of the objective function have a long, narrow valley or ridge. The eigenvalues of the Hessian matrix reveal the presence of such features in the contours. Specifically if there exists a small eigenvalue then the contours will be elongated in the direction of the corresponding eigenvector.

The Newton direction can be determined if we perform the spectral decomposition of \mathbf{H}

$$\mathbf{H} = \mathbf{V} \Lambda \mathbf{V}^T$$

where \mathbf{V} is the matrix of eigenvectors and $\mathbf{\Lambda}$ the diagonal matrix of eigenvalues. Now \mathbf{H} is symmetric so the matrix of its eigenvectors is unitary i.e. $\mathbf{V}^T = \mathbf{V}^{-1}$, and

$$\mathbf{H}^{-1} = \mathbf{V}\mathbf{\Lambda}^{-1}\mathbf{V}^T$$

\mathbf{V} forms a basis for this vector space so we can write the gradient as a linear combination of the eigenvectors $\mathbf{g} = \mathbf{V}\mathbf{\alpha}$, and from Eq. 2.5 obtain the Newton step as

$$\begin{aligned}\mathbf{H}^{-1}\mathbf{g} &= \mathbf{V}\mathbf{\Lambda}^{-1}\mathbf{V}^T\mathbf{V}\mathbf{\alpha} \\ &= \mathbf{V}\mathbf{\Lambda}^{-1}\mathbf{\alpha} \\ &= \begin{bmatrix} \frac{v_1}{\lambda_1} & \frac{v_2}{\lambda_2} & \frac{v_3}{\lambda_3} & \dots & \frac{v_n}{\lambda_n} \end{bmatrix} \mathbf{\alpha}\end{aligned}$$

Now if the eigenvalue λ_j is very small, then unless α_j is also coincidentally small, the Newton step will have a large component in the direction of the eigenvector v_j . So the method makes long strides along valleys or ridges and avoids the *hemstitching* of a steepest descent type method.

While Newton's method has these advantages, practical problems occur if it is used in an unmodified form.

- The Newton direction is not guaranteed to be a descent direction when far from the optimum. In its unmodified form, Newton's method may diverge and stop at saddle points or maxima. The presence of negative eigenvalues indicates that Newton's method will move towards such a point.
- Even though the long steps along ridges and valleys are potentially desirable it may not always be advisable to take these long steps since the quadratic model is only a local approximation of the objective function.
- Furthermore, since the function does not change much in this direction (i.e. along the valleys or ridges) it may be undesirable to move too far along it for the small resulting decrease in the objective function.
- Small eigenvalues make \mathbf{H} ill-conditioned, so numerical solution of Eq. (2.5) may produce a poor result.
- Second partial derivatives are needed which are expensive to compute and complicated to evaluate.

2.2. THE NONLINEAR LEAST-SQUARES PROBLEM

For nonlinear least squares the objective function is the sum of squares of the residuals and has a structure which allows for special treatment to avoid some of the problems of Newton's method:

$$\mathbf{F} = \frac{1}{2} \mathbf{f}^T(\mathbf{x}) \mathbf{f}(\mathbf{x}) \tag{2.7}$$

where

$$\mathbf{f}(\mathbf{x}) = [\mathbf{f}_i] = \mathbf{F}(\mathbf{x}, \mathbf{t}_i) - y_i \quad (2.8)$$

Here (\mathbf{t}_i, y_i) are a set of m observations of an independent and dependent variable (typically time and pressure in a well test) and \mathbf{x} is the set of n unknown reservoir parameters. The reservoir response function \mathbf{F} is a nonlinear function of the unknowns \mathbf{x} and time \mathbf{t} .

For this objective function the gradient is

$$\mathbf{g}(\mathbf{x}) = \mathbf{J}(\mathbf{x})^T \mathbf{f}(\mathbf{x}) \quad (2.9)$$

where $\mathbf{J}(\mathbf{x})$ is the $m \times n$ Jacobian matrix:

$$\mathbf{J}(\mathbf{x}) = [\mathbf{j}_{ij}] = \left[\frac{\partial f_i}{\partial x_j} \right] \quad (2.10)$$

while the Hessian is

$$\mathbf{H}(\mathbf{x}) = \mathbf{J}(\mathbf{x})^T \mathbf{J}(\mathbf{x}) + \sum_{i=1}^m f_i(\mathbf{x}) \mathbf{G}_i(\mathbf{x}) \quad (2.11)$$

where \mathbf{G}_i is the $n \times n$ matrix of second partial derivatives at the observation point \mathbf{t}_i

$$\mathbf{G}_i(\mathbf{x}) = [\mathbf{g}_{jkl}] = \left[\frac{\partial^2 f_i}{\partial x_j \partial x_k} \right] \quad (2.12)$$

So the Hessian matrix has a combination of first and second partial derivatives. One of the objectives of this study is to determine the relative merits of omitting or retaining the second order derivatives.

2.3. DESCRIPTION OF THE PROCEDURES

This section discusses the properties of some of the common nonlinear regression methods and the method under study. For a broader view of nonlinear parameter estimation methods, see Bard (1974) and Gill, Murray and Wright (1983).

All of the methods used here and elsewhere to improve on Newton's Method are aimed at guaranteeing that the solution is always moving *downhill*, which is usually achieved by modifying \mathbf{H} in Eq. 2.5 such that it becomes positive-definite, (i.e. all eigenvalues positive). Four methods are discussed: Method of Steepest Descent, Gauss Method, Marquardt Method and Greenstadt Method.

2.3.1. The Method of Steepest Descent

This method replaces \mathbf{H} in Eq. 2.5 by the identity matrix \mathbf{I} multiplied by a scalar λ . In this way the amount by which \mathbf{p}_i is changed in each iteration is directly proportional to the

gradient of F with respect to that parameter. Thus the solution moves *downhill* most steeply in the direction of the parameter which most reduces F , hence the name *steepest descent*. The method of steepest descent is very simple, however it is slow to converge and often requires hundreds of iterations to reach the minimum. Its performance is particularly bad when the contours of the objective function resemble a long narrow valley. While Newton's method makes strides along the axis of the valley, steepest descent will move repeatedly back and forth across the valley (i.e., *hemstitching*) in the manner of a heavy ball let loose on the side of a narrow valley. Progress along the axis of the valley will be very slow.

2.3.2. The Gauss Method

This method takes advantage of the least-squares origin of the objective function and treats the second derivatives in Eq. 2.11 as if they were zero. So $H(x) = J(x)^T J(x)$, which is always positive-definite. In addition, since it is no longer necessary to evaluate the second derivatives, there is a considerable savings both in computational time and in the algebra needed to obtain expressions for the elements of the Hessian.

The method requires the solution of the normal equations at the k th iteration

$$J_k^T J_k p_k = -J_k^T f_k \quad (2.13)$$

These equations are similar to those that arise in linear least-squares problems. The Gauss method can thus be thought of as sequence of linear least-squares problems. One difference is that in nonlinear least-squares the equations need not be solved as accurately, for each solution is just a step in an iterative process towards the minimum.

The Gauss method often converges as fast as the Newton method (which can be shown to be the most efficient in convergence as the optimum is approached (Greenstadt, 1967)). Note that as the reservoir model F approaches the observations y , the term in Eq. 2.11 that is ignored in the Gauss method approaches zero even in the Newton method. Hence the Gauss and Newton method become approximately the same provided the model really is a close match to the data so that the residual is small at the optimum. Gill et al. (1983) point out that ignoring the second order terms is not justified when the residual at the optimum $\|f(x^*)\|_2$ is comparable to the largest eigenvalue of $J(x^*)^T J(x^*)$.

In theory the Gauss method should always converge to a minimum because it always moves in a descent direction. However in practice, it is sensitive to computational instability which corresponds to a near singular Hessian matrix H . This occurs partly because the underlying mathematical model is ill-defined, unfortunately well test analysis applications are often prone to this kind of difficulty (Rosa and Horne, 1983), and partly because the condition number of J^T is the square of the condition number of J .

2.3.3. The Marquardt Method

The Marquardt (1963) method is a modification that can be made to either the Gauss or Newton methods, although it is commonly used with the Gauss method. In this method a constant, μ , is added to the diagonal elements of H . This is equivalent to adding μ to the eigenvalues of H .

In the case of the Newton method, μ must be chosen such that any negative eigenvalues of the original matrix \mathbf{H} become positive, thus ensuring positive-definiteness.

With the Gauss method the eigenvalues are already always non-negative, however the addition of the Marquardt parameter ensures that no zero or almost zero values occur, thus preventing computational singularity of the matrix. For the k th iteration of the Gauss-Marquardt method the normal equations become:

$$(\mathbf{J}_k^T \mathbf{J}_k + \mu_k \mathbf{I}) \mathbf{p}_k = -\mathbf{J}_k^T \mathbf{f}_k \quad (2.14)$$

The condition number is now much improved for $(\lambda_{\max} + \mu) / (\lambda_{\min} + \mu) < (\lambda_{\max}) / (\lambda_{\min})$. As the iterations proceed the Marquardt parameter μ is reduced by a factor of 10 in each iteration where extrapolation (increase in size of step ρ) is possible.

The addition of μ on the diagonal of the solution matrix makes the method act in a fashion similar to the method of steepest descent in the early iterations. Thus when far from the optimum the Marquardt method benefits from the initial rapid reduction of \mathbf{F} characteristic of steepest descent, yet converges like Newton's method as the optimum is approached. Thus, in this and earlier studies, the Gauss-Marquardt method was often found to converge faster than Newton's method.

After several iterations μ becomes very small, and the method will approach the original method (either Gauss or Newton).

The Marquardt method sometimes runs into difficulties. The Marquardt parameter μ is increased by a factor of 10 in each iteration where interpolation is needed to obtain a decrease in the objective function. When passing through particularly ill-conditioned regions, the parameter may become quite large.

In such cases the method once again behaves in the manner of steepest descent but this tends to converge slowly in such regions. Moreover when μ is large the Marquardt step also becomes very small, for:

$$(\mathbf{H} + \mu \mathbf{I}) \mathbf{p} = -\mathbf{g}$$

may be written as

$$\left[\frac{1}{\mu} \mathbf{H} + \mathbf{I} \right] \mathbf{p} = -\frac{1}{\mu} \mathbf{g}$$

so as $\mu \rightarrow \infty$, the step becomes an infinitesimal one in the steepest descent direction. Theory suggests taking the Marquardt step always, but in practice one also uses a line search procedure to adjust the step length. However, due to the poor steepest descent direction, the step size will likely remain small so the method can get *stuck* taking small steepest descent steps in ill-conditioned regions.

2.3.4. The Newton-Greenstadt Method

This method is another way to guarantee positive-definiteness of the Hessian matrix. Greenstadt (1967) proposed performing the spectral decomposition of \mathbf{H} to reveal the presence

of zero or negative eigenvalues, and then using the absolute value of the eigenvalues before proceeding with the calculation of the inverse. If one of the eigenvalues is zero or close to zero, this means that the model is insensitive to one of the parameters or to one particular combination of the parameters. Greenstadt has proposed that any zero or almost zero eigenvalue be replaced by an infinity; this is achieved by replacing the reciprocal of the eigenvalue by zero when calculating the inverse. Thus any parameter which does not affect the value of F will remain unchanged. This is what is desired in an automated well test analysis and is the opposite of what will occur with Newton's method in which the least sensitive parameters are changed most (by the long strides along valleys).

Thus the Newton-Greenstadt method should provide good performance for automated well test analysis applications, which require an algorithm that is robust, rapidly convergent, and which is not influenced by parameters that might not affect the reservoir model response for a particular set of data. This study was performed to determine whether the Newton-Greenstadt method has the desired properties in practical application, and whether the additional effort required to calculate the second derivatives and the eigenvalues results in a performance advantage.

2.3.5. Penalty Functions

All the methods can use penalty functions [Bard (1974), Rosa and Horne (1983)] to prevent the search direction from approaching infeasible regions. The penalty functions provide a simple means of adding *inequality* constraints to the problem without having to change from unconstrained optimization techniques to constrained techniques. To impose the bounds $a_\alpha \geq x_\alpha \geq b_\alpha$, we use the two constraints

$$h_j \equiv a_\alpha - x_\alpha$$

$$h_{j+1} \equiv x_\alpha - b_\alpha$$

Such constraints are incorporated for each parameter as *penalty functions* added to the objective function

$$F \leftarrow F + \sum_j \frac{\alpha_j}{h_j}$$

The constant α_j is small so the penalty functions normally have little effect on the convergence. In addition α_j can be progressively decreased so as to have negligible effect near the optimum--a simple way to do this is to let α_j decrease with F . However when the search process approaches the limits a_α or b_α the corresponding constraint h_j tends to zero. The penalty function α_j/h_j then becomes large and deflects the search from the infeasible direction. Since the penalty functions are now a part of the objective function, the first and second partial derivatives of α_j/h_j with respect to x_α must also be included in gradient and Hessian respectively.

2.3.6. Line Search

The line search procedure used in all of the algorithms is described in Bard (1974). It works by first making a trial step length ρ_0 . It then fits a one dimensional quadratic function

through the trial point $\mathbf{x} + \rho_0 \mathbf{p}$ and the current point \mathbf{x} . The quadratic is obtained from these two points alone for if we define a function

$$\psi(\rho) \equiv F(\mathbf{x} + \rho \mathbf{p})$$

its derivative is

$$\begin{aligned} \frac{\partial \psi}{\partial \rho} &= \left[\frac{\partial F}{\partial \mathbf{x}} \right]^T \mathbf{p} \\ &= \mathbf{g}^T \mathbf{p} \end{aligned}$$

At the k th iteration we know both \mathbf{g}_k and \mathbf{p}_k , so $\partial \psi / \partial \rho$ can be found. We also know $\psi(0) = F(\mathbf{x})$ and $\psi(\rho_0) = F(\mathbf{x} + \rho_0 \mathbf{p})$. With these three values the three coefficients of the quadratic $a\rho^2 + b\rho + c$ can be obtained. In this way a quadratic model of the objective function is obtained in the direction of the step vector, and we can proceed directly to the minimum of this quadratic using $\rho = -b/2a$.

Since the quadratic may not closely model the function, the algorithm uses certain heuristics to guide further interpolation-extrapolation steps and the reader is referred to Bard (1974) for details.

2.3.7. Solving the Matrix Problem

The Newton-Greenstadt method requires the spectral decomposition of the Hessian matrix. In practical applications, scaling of the Hessian matrix is first done to improve the condition number. Bard (1974) suggests a simple scaling that reduces all diagonal elements to unit magnitude. This amounts to rescaling all variables so that the curvature of the objective function at the minimum is unity along all coordinate axes. The proposed scaling makes all off-diagonal elements less than unity except when the matrix is indefinite. This happens with Newton's method in the early iterations but experiments show that this does not affect the performance of the nonlinear method i.e., steps sizes and directions remain acceptable. Bard (1974) also reports that the method has given very good results.

The *inverse scaled decomposition* thus provides a numerically accurate and stable means of applying the Greenstadt modification and obtaining the inverse of the matrix. For the matrix \mathbf{H} this is obtained by the following steps:

1. Divide each element H_{ij} by $|H_{ij}H_{jj}|^{1/2}$, forming the matrix \mathbf{B} . All zero H_{jj} are replaced by one before performing this scaling.
2. Obtain the eigenvalues λ_i and eigenvectors \mathbf{u}_i of \mathbf{B} .
3. Divide the j th element of \mathbf{u}_i by $|H_{jj}|^{1/2}$ to form the vector \mathbf{c}_i , the i th column of \mathbf{C} .
4. The inverse scaled decomposition of \mathbf{H} is given by $\mathbf{H}^{-1} = \mathbf{C}\mathbf{\Lambda}^{-1}\mathbf{C}^T$.

This scaling is useful also for Newton and Gauss-Marquardt methods. The Greenstadt modification consists of adjusting the eigenvalues prior to forming $\mathbf{\Lambda}^{-1}$. The method is applied in two ways, in the traditional variation negative eigenvalues are replaced by their absolute values to ensure a descent direction. This will not solve the problem of ill-conditioning because

zero or near zero eigenvalues need to be handled too. So in the second variation all eigenvalues less than a given number (this includes all negative eigenvalues) are replaced by a large positive number. The latter modification serves to both ensure a positive-definite Hessian and to control the effect of ill-defined parameters on the search. For simplicity in nomenclature we refer to these as method (a) and method (b) respectively. Method (b) was implemented by replacing all eigenvalues less than 0.00001 by 5.

2.4. EVALUATION OF THE ALGORITHMS

Several types of well test were analyzed using the different estimation algorithms, and two representative examples are illustrated here. The first is the field example analyzed by Rosa and Horne (1983). The test is a drawdown in a homogeneous reservoir with storage and skin at the producing well. Several runs were made on this data and these are referred to as Case 1 through Case 5. The second test is a simulated test on a dual-porosity reservoir. Reservoir parameters for the two test cases are listed in Tables 2.1 and 2.3. Figure 2.1 shows the first test data with a typical match drawn as a solid line. The response without storage is also drawn as a solid line to show the extent of semilog straight line data available for a manual match.

The parameters normally sought in this type of test are permeability, storage and skin. In addition, the porosity-compressibility product was also used as a parameter in some of the tests. The experience of Gringarten *et. al.* (1979) and Rosa and Horne (1983) has shown the porosity-compressibility product to be a very ill-defined parameter that cannot be uniquely determined with a graphical type-curve match or by using the Gauss-Marquardt automated matching procedure. The inclusion of this parameter is therefore a stringent test of the automated methods. The initial guesses used in the analysis of the first test are listed in Table 2.2. Those for the second test are listed in Table 2.4.

2.4.1. Storage and Skin, Case 1

Figure 2.2 shows the convergence obtained with 4 different estimation techniques. The three standard parameters (k , s and C) were to be determined and the initial guess is a good one. Figure 2.3 shows that the Newton method converges fastest of all and much faster than Gauss-Marquardt. It appears that the second order information available in the Newton method can lead to a dramatic increase in performance but, as will be shown later, this is not usually the case. We also see that the (b) variation of Newton-Greenstadt is nearly as good as Newton's method itself.

Figure 2.3 shows the variation of the eigenvalues during the iterative process of Newton's method. In the early stages one of the eigenvalues is a small negative number. This illustrates one of the problems with Newton's method - the Hessian matrix is not guaranteed to be positive-definite.

In this case the two positive eigenvalues indicate progress toward the minimum in two directions while the negative eigenvalue indicates progress towards the maximum in the third direction.

The (a) variation of the Newton-Greenstadt method corrects this kind of problem by changing the sign of the negative eigenvalues and in effect *turns around* the search so that it is facing downhill in all directions.

Table 2.1. Data for storage and skin tests

			Δt , hrs	Δp , psi
			0.0167	26.
			0.033	53.
			0.05	78.
			0.1	148.
			0.15	210.
			0.2	267.
			0.3	369.
			0.4	454.
			0.5	519.
			0.6	572.
Formation thickness	69.00	ft	0.8	654.
Oil Viscosity	0.90	cp	1.0	719.
Wellbore Radius	2.75	in	1.5	804.
Oil Production Rate	333.95	RB/D	2.0	849.
			2.5	859.
			3.0	864.
			3.5	869.
			4.0	874.
			4.5	877.
			5.0	882.
			6.0	887.
			7.0	893.
			8.0	897.
			10.0	905.
			12.0	911.

Table 2.2. Initial guesses, bounds and best-fit values for storage and skin test are listed in Table 2.4

	Permeability (darcies)	Skin	Storage (bbl/psi)	ϕc_i (1/psi)
Case 1	0.01	7	0.001	-
Case 2	0.01	7	0.001	0.1613e-6
Case 3	0.001	1	0.000558	-
Case 4	0.001	1	0.000558	0.2613e-6
Minimum	0	0	0.0	0.1e-8
Maximum	1	30	2.6	0.1e-4
Best-fit	0.0155	14.9	0.0085	0.2613e-6

Table 2.3. Data for double porosity tests

			Δt , hrs	Δp , psi
			0.01	46.
			0.02	74.
			0.03	91.
			0.04	102.
			0.06	113.
			0.08	119.
			0.10	121.
Formation thickness	13.00	ft	0.3	126.
Oil Viscosity	0.30	cp	0.5	127.
Wellbore Radius	3.48	in	0.7	128.
Oil Production Rate	1245.00	RB/D	1.0	129.
			4.0	136.
			7.0	139.
			10.0	140.
			15.0	142.
			19.0	144.
			24.0	145.

Table 2.4. Initial guesses, bounds and best-fit values for double porosity test

	Permeability (darcies)	Skin	Storage (bbl/psi)	ϕc_t (1/psi)	ω	λ
Guesses used for Double-Porosity Cases						
5 parameters	0.152	-1.1	0.001	-	0.36	0.8 e-8
6 parameters	0.152	-1.1	0.001	0.1 e-6	0.36	0.8 e-8
Guesses used for Storage and Skin Case 5						
Case 5	0.01	7	0.001	-	0	1
Inequality Bounds for Double-Porosity Test						
Minimum	0	-30	0.0	0.1e-8	0.001	.1 e-10
Maximum	1	30	0.6	0.1e-4	5	.1 e-4
Inequality Bounds for Storage and Skin Case 5						
Minimum	0	-30	0.0	0.1e-8	0	0
Maximum	1	30	0.6	0.1e-4	1	1
True values	0.452	4.1	0.0085	0.1 e-6	0.16	2.8 e-8

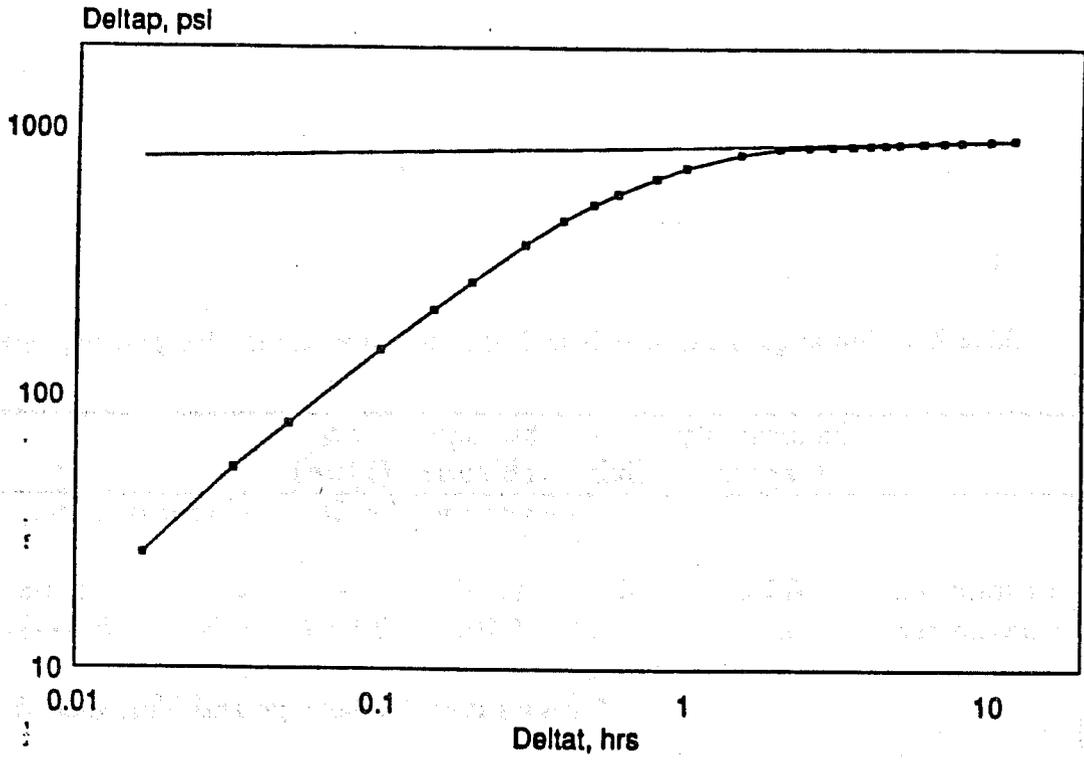


Figure 2.1. Storage and skin test.

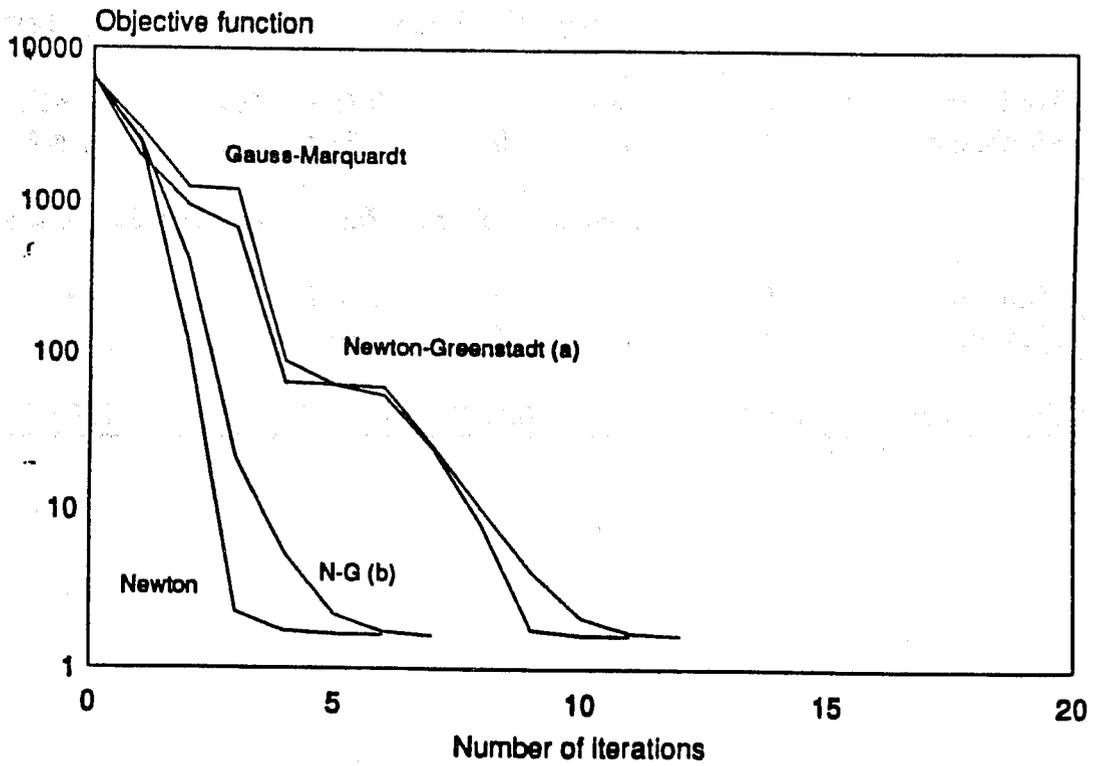


Figure 2.2. Convergence, Case 1.

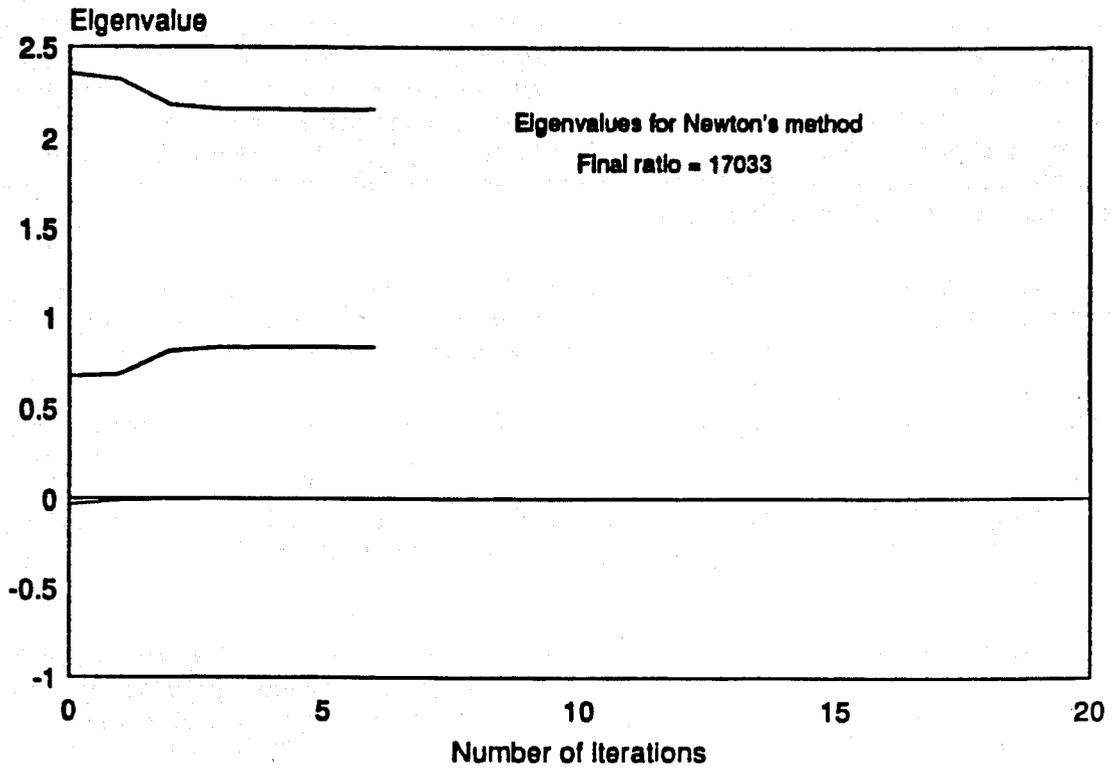


Figure 2.3. Eigenvalues of Newton's method, Case 1.

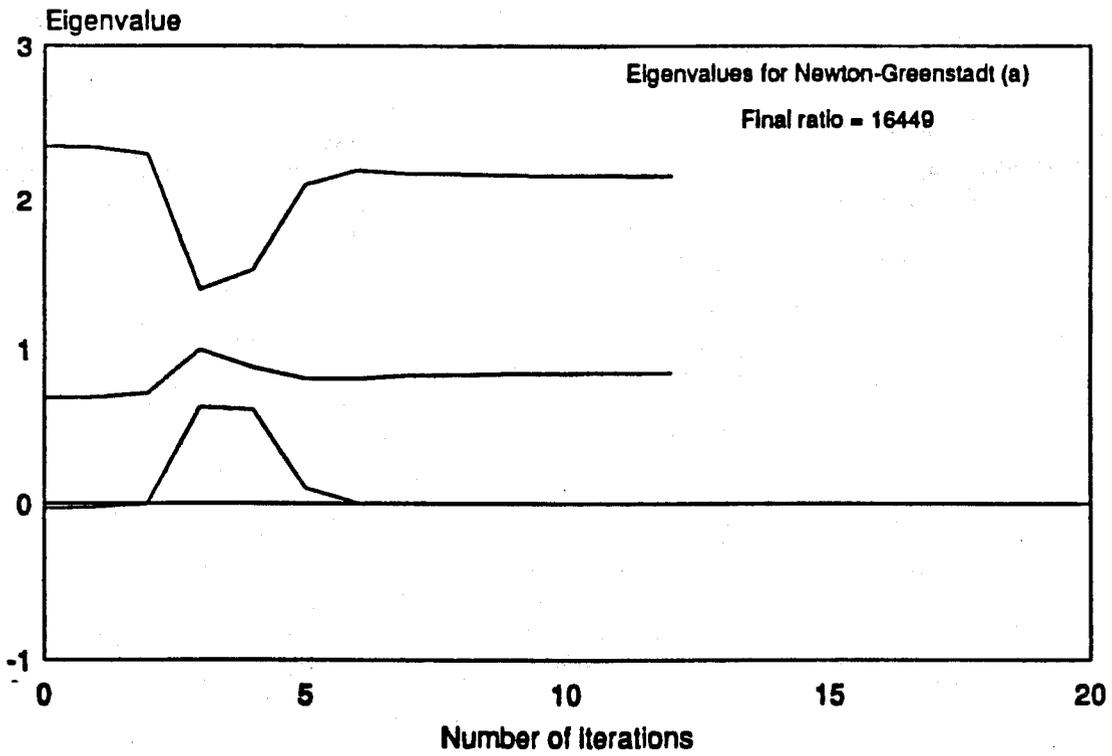


Figure 2.4. Eigenvalues of Newton-Greenstadt method, Case 1.

The convergence shown in Fig 2.2 shows that despite the negative eigenvalue during the first iterations, Newton's method still can go on, in this particular case, and find the true minimum of the objective function. Though the algorithm is trying to maximize in the direction of the negative eigenvalue, its magnitude is small so little harm is done because the function increases only slowly in that direction. Since it is at the same time minimizing in the other two directions, Newton's method progresses towards the global minimum. Eventually the small eigenvalue changes sign and becomes positive from the third iteration onward and then minimization occurs in all three directions.

So even without a positive-definite Hessian matrix Newton's method has managed to converge in this case. However we expect that when the negative eigenvalue is large in magnitude Newton's method will run into trouble, and this happened in later tests.

Figure 2.4 shows the eigenvalues for the (a) variation of Newton-Greenstadt, prior to applying Greenstadt's modification. Note that although the method has simply turned around one of the eigenvalues, this has affected the entire search process and convergence requires 12 iterations compared to 6 iterations for Newton's method. In the (b) variation we are stopping or nearly stopping any movement in the direction of the negative eigenvalue and this method converges in 7 iterations as shown in Figure 2.2.

The eigenvalues for the Gauss method are always positive although they may often be very small. Figure 2.5 shows the eigenvalues for the Gauss method prior to applying Marquardt's modification and we see that at certain points there is one small eigenvalue. The final eigenvalue ratio of the Hessian (i.e., of the product $\mathbf{J}^T\mathbf{J}$) is 16137, indicating some ill-conditioning, but obviously the Marquardt method can successfully handle this problem. Both the eigenvalue and convergence plots show a striking similarity between the Newton-Greenstadt (a) and the Gauss-Marquardt method; this is a coincidence since they usually converge very differently.

2.4.2. Storage and Skin, Case 2

Using the same (good) initial guesses for \mathbf{k} , \mathbf{s} and \mathbf{C} as in Case 1, the additional parameter $\phi\mathbf{c}_t$ was also sought as an unknown in this test. Figure 2.6 shows that the Gauss-Marquardt method is seriously affected by the addition of this new, ill-conditioned parameter. While the performance of all the second order methods has remained nearly the same, the Gauss-Marquardt method is converging only very slowly.

This is the kind of behavior that prompted this study. It is not uncommon in particularly complicated models for the Gauss-Marquardt method to spend scores of iterations in steepest descent type progress. This was handled in previous studies (Barua *et al.* 1985) by first doing a manual semilog analysis to find \mathbf{k} and \mathbf{s} . The two parameters can be held fixed at these values. Once the number of unknowns is reduced in this way convergence quickly follows. However, one of the benefits of automated type curve match is that we can avoid choosing the wrong straight line, but this advantage is lost if we are forced to do so for the sake of obtaining convergence.

Another effect to note in Fig. 2.6 is that *both* second order methods are unaffected by the addition of the ill-defined parameter. This indicates that second order information does provide some benefit when dealing with ill-defined parameters. Later tests also show the same effect.

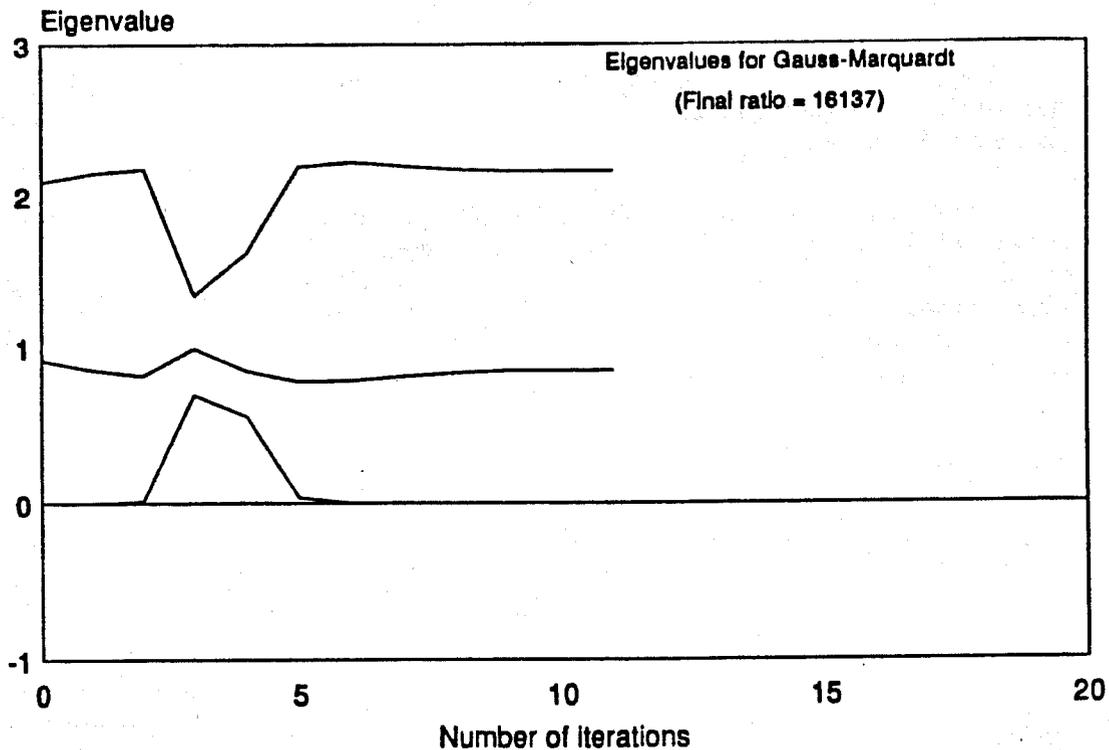


Figure 2.5. Eigenvalues of Gauss-Marquardt method, Case 1.

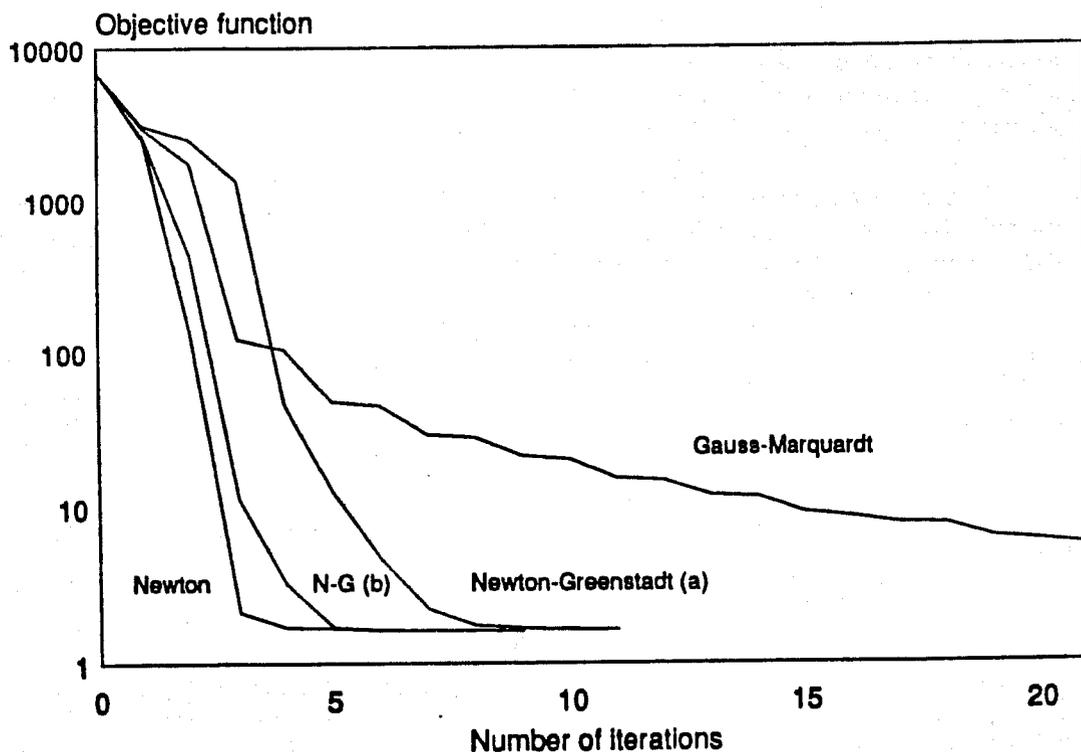


Figure 2.6. Convergence, Case 2.

Figure 2.7 shows the eigenvalues for the Gauss method prior to Marquardt's modification. While there was just one small eigenvalue in Case 1, there now are two small eigenvalues, probably because skin and ϕc_t are highly correlated.

The ratio of the maximum and minimum eigenvalues at the solution provides the condition number of the Hessian and this ratio is commonly used as a measure to see whether a problem is ill-defined.

The final eigenvalue ratio in Fig. 2.7 is 37449, which is not very much larger than the ratio of 16137 in Fig. 2.5. The performance is so greatly affected in this case that it leads us to believe that the presence of more than one very small eigenvalue is more troublesome than a high eigenvalue ratio. In other words multiple rank deficiency of the Hessian matrix is likely to be more troublesome than simple ill-conditioning of the matrix. This observation is confirmed in later tests.

2.4.3. Storage and Skin, Case 3

While the previous two cases showed second order methods converging faster than the Gauss-Marquardt method, they usually do not do so. Unless the Gauss-Marquardt method gets *stuck*, tests showed that it generally converges faster than a second order method on the relatively noise-free data used for the test cases.

Figure 2.8 shows the results of estimating k , s and C with a different set of initial guesses that are far from the solution (see Table 2.2). This figure shows the Gauss-Marquardt converging much faster than any other method. Also Newton's method fails after 6 iterations.

Figure 2.9 shows the eigenvalues for Newton's method and we can see the same problem observed earlier, an indefinite Hessian causing maximization in one direction. This time the negative eigenvalue is large in magnitude so the function value is increasing by a large amount in the direction of the corresponding eigenvector. In spite of this, the method is able to achieve an overall reduction in the objective function over the first 6 iterations, but fails thereafter.

Figure 2.9 also shows why Marquardt's modification is not very beneficial for Newton's method. To make all eigenvalues positive requires a Marquardt parameter orders of magnitude larger than normal in the Gauss-Marquardt method. Since a large Marquardt parameter restricts the size of the step and forces steepest-descent type behavior, very poor progress results from this. This was confirmed by experiments. Figure 2.10 shows the eigenvalues for the Newton-Greenstadt (a) prior to modification. From the second iteration onward all the eigenvalues are positive so the method reduces to Newton's method thereafter. It is obvious that in spite of using second order information without any compromises to ensure positive-definiteness the convergence is still not as good as for Gauss-Marquardt.

Figure 2.11 explains why the convergence is so poor. Newton's method assumes that the objective function can be described by a locally quadratic model function. If this is so, then a Newton step of unity takes us to the best local minimum. Newton's method is at its most efficient and converges quadratically only when the step length is unity. Figure 2.11 shows that except for one occurrence nowhere else has the step length of unity been chosen. So even though second order information is used, it is inadequate to describe the function exactly, so the expected rate of convergence does not materialize. It is apparent that a quadratic model will not be able to closely approximate the local behavior of the objective function in well test

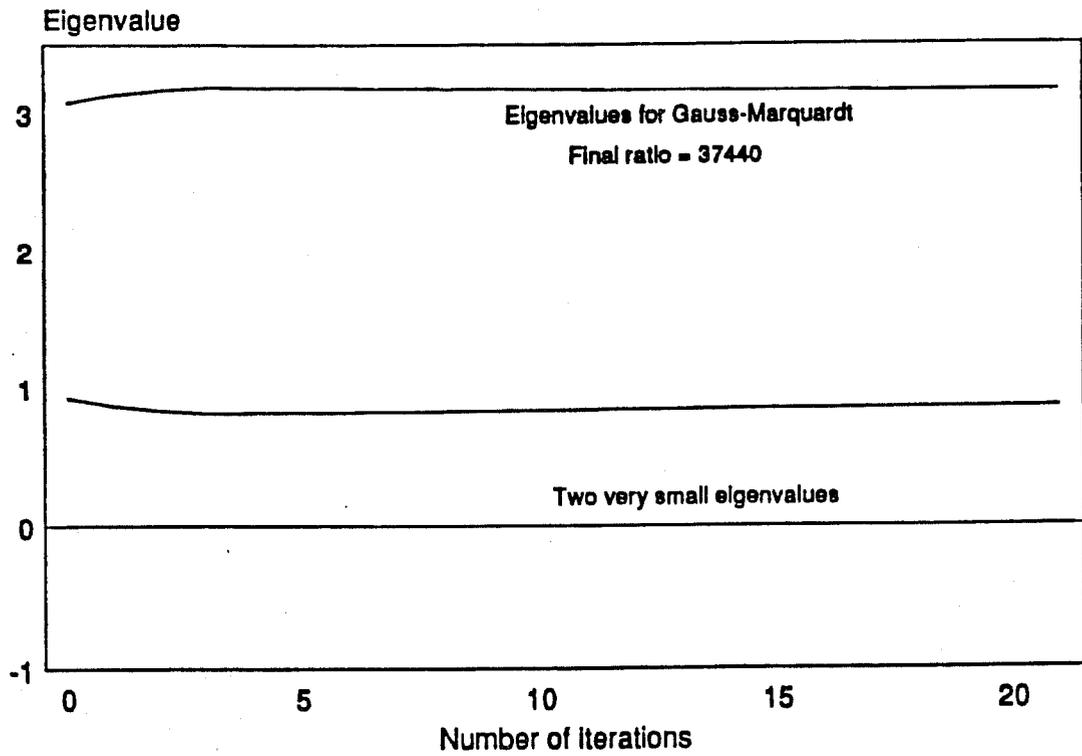


Figure 2.7. Eigenvalues of Gauss-Marquardt method, Case 2.

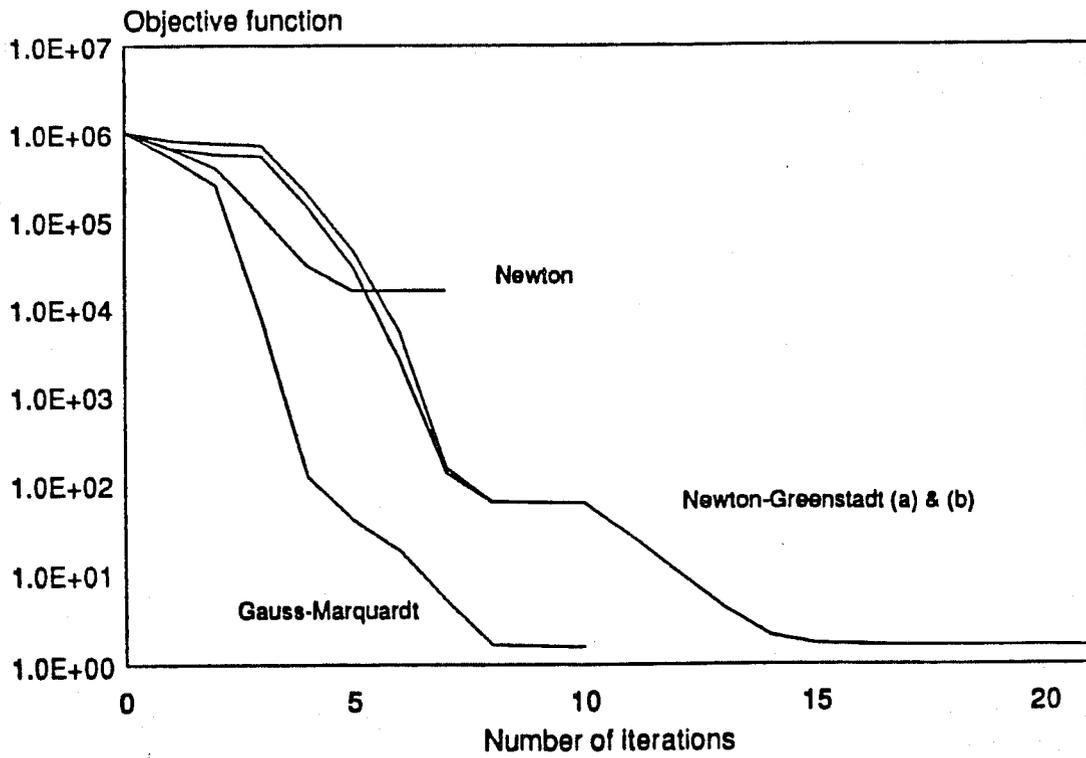


Figure 2.8. Convergence, Case 3.

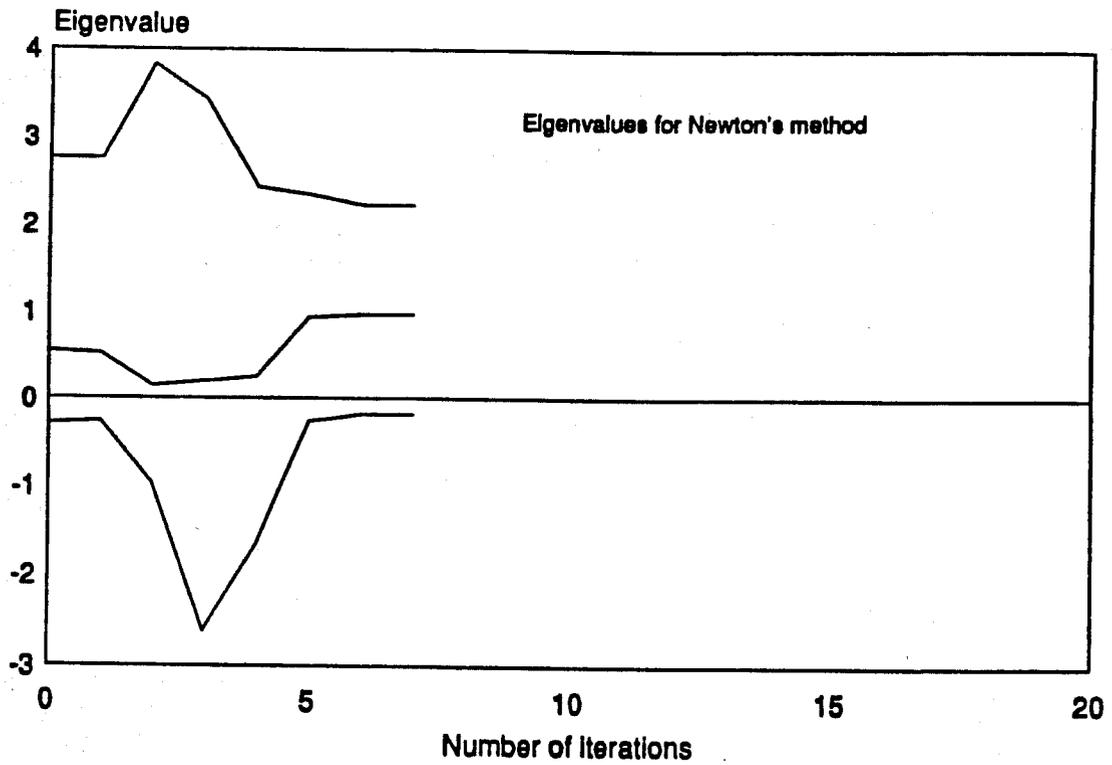


Figure 2.9. Eigenvalues of Newton's method, Case 3.

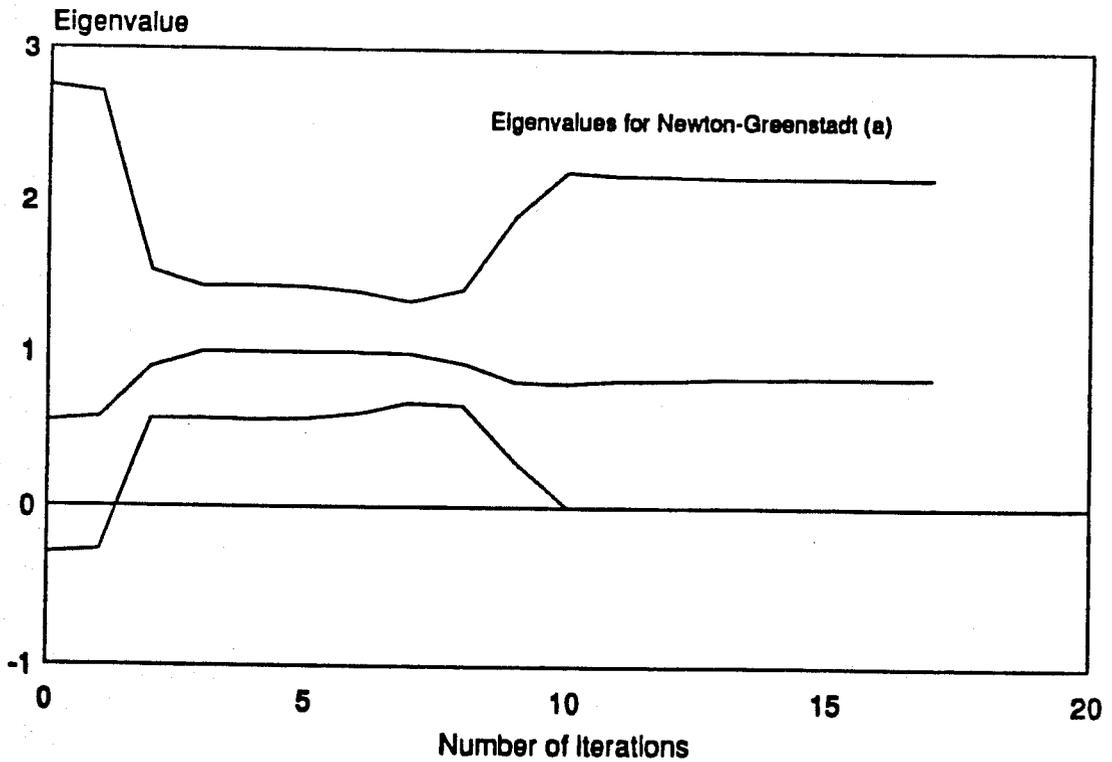


Figure 2.10. Eigenvalues of Newton-Greenstadt method, Case 3.

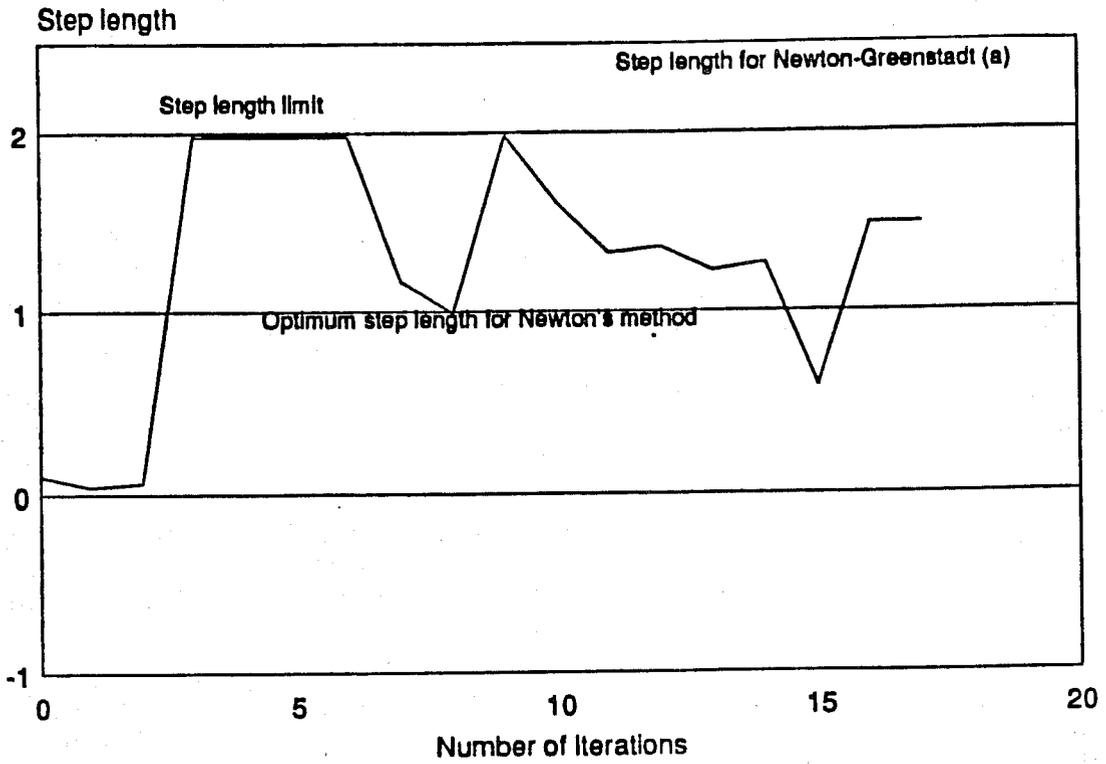


Figure 2.11. Step lengths of Newton-Greenstadt (a) method, Case 3.

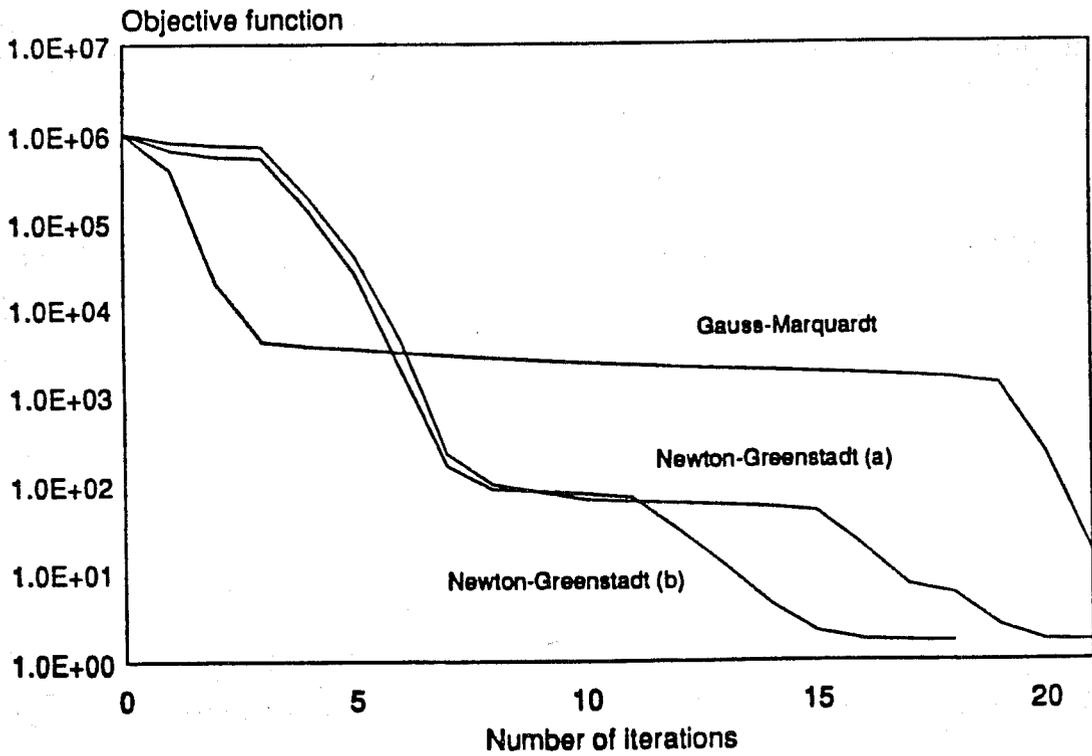


Figure 2.12. Convergence, Case 4.

analysis at least when far from the optimum. Therefore we do not expect quadratic convergence from Newton's method. In fact as Fig. 2.8 shows, the convergence may actually be slower when using second derivative information.

2.4.4. Storage and Skin, Case 4

This test uses the same (poor) initial guesses as Case 3, with ϕc_t also sought as an unknown with its initial guess left exactly at the true value. Figure 2.12 shows the convergence for the different methods and here again the effects noted in Case 2 appear. The Gauss-Marquardt method is badly affected by the introduction of the fourth parameter.

Second order methods fare better. Newton-Greenstadt (a) is affected by the introduction of the ill-defined parameter but less so than Gauss-Marquardt. Newton-Greenstadt (b) performs about the same as when there were only three parameters.

So second order methods do seem to provide some help in the case of ill-defined parameters but restricting the magnitude of the step in ill-defined directions [as in NG(b)] is even more helpful. This has been confirmed recently by Nanba and Horne (1988) who used a scheme similar to NG(b) on a Modified Cholesky Gauss method.

Since the Newton-Greenstadt (b) modification has successfully turned an otherwise fragile method into a more robust one we wonder if it can do the same thing for the Gauss-Marquardt method. Figure 2.13 shows the result of such an attempt. In this test any eigenvalue less than 0.001 was set to 5 before the Marquardt modification was made to the remaining eigenvalues. We see that this modification has only a small effect. It is also not easy to determine the appropriate minimum eigenvalue to use in this case. Too small a cutoff leaves the problem unchanged, while too large a cutoff may affect more than one value and continue to have an effect even near the solution. In contrast the choice of 0.00001 as a cutoff for the second order method was made relatively easily since it is only desired to affect negative eigenvalues and eigenvalues so small that they have virtually no effect on the function.

2.4.5. Double-Porosity Case

Double-porosity well tests are of considerable interest given their complexity in manual interpretation. Figure 2.14 shows a simulated well test. Note that there are only a few data points and a small amount of *noise* introduced by truncating all pressure values to integers. Table 2.3 lists the test data while Table 2.4 lists the initial guesses used in the estimation process.

Figure 2.15 shows the convergence for the 5 parameters (k , s , C , ω and λ). Note that even though there are more parameters than before, still the Gauss-Marquardt method has converged fastest. Newton's method failed after one iteration while the Newton-Greenstadt (b) is second best. Notice the very poor performance of NG (a) with its unrestricted step sizes along the valleys.

Figure 2.16 shows the eigenvalues for the Gauss-Marquardt method. Here only one eigenvalue is small and the same rapid convergence observed earlier is apparent. So a large number of parameters does not necessarily mean difficulties for the Gauss-Marquardt method.

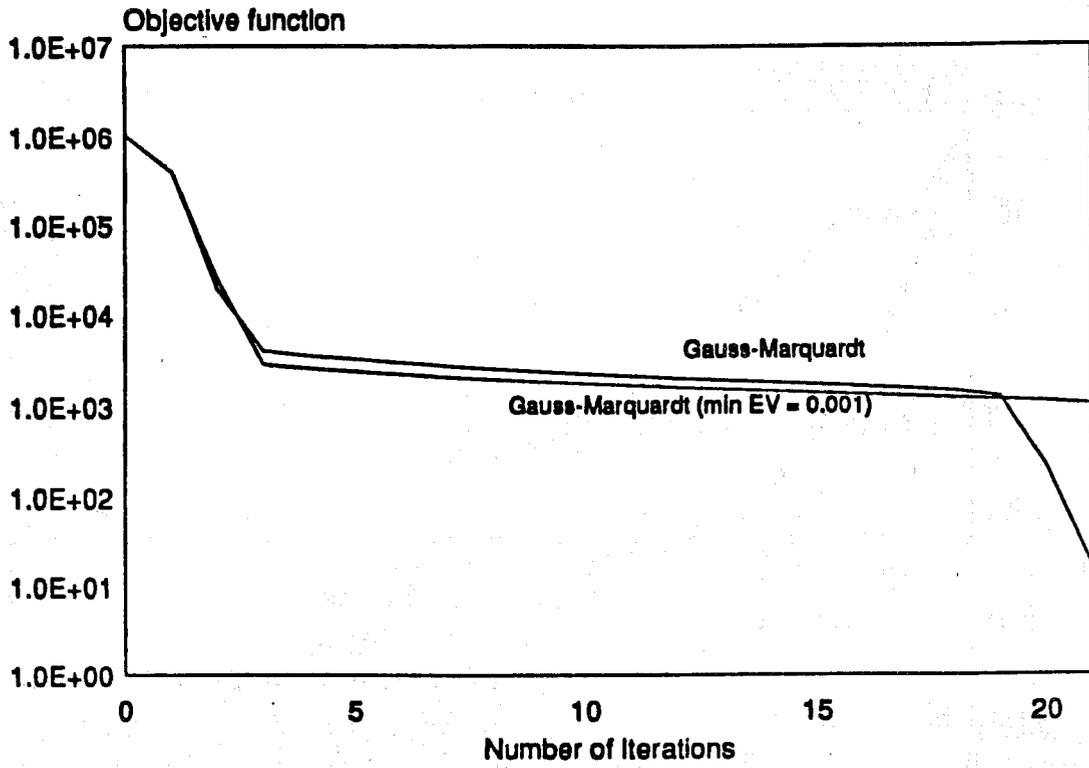


Figure 2.13. Gauss-Greenstadt, Case 4.

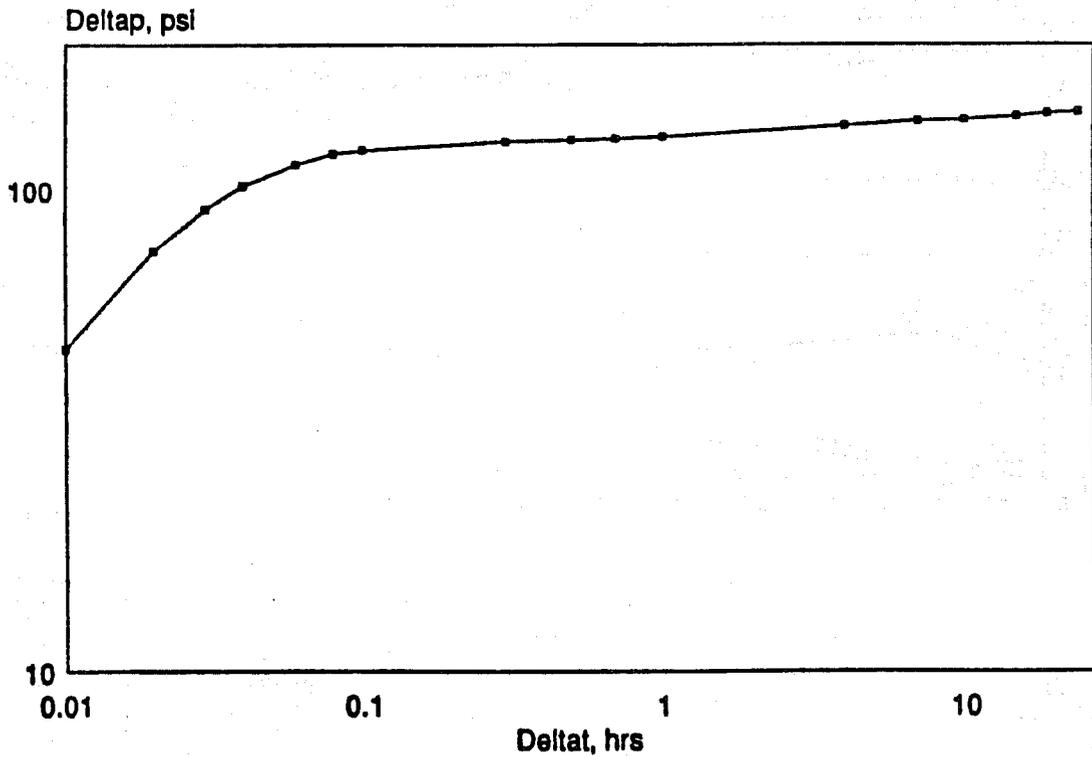


Figure 2.14. Double-porosity test case.

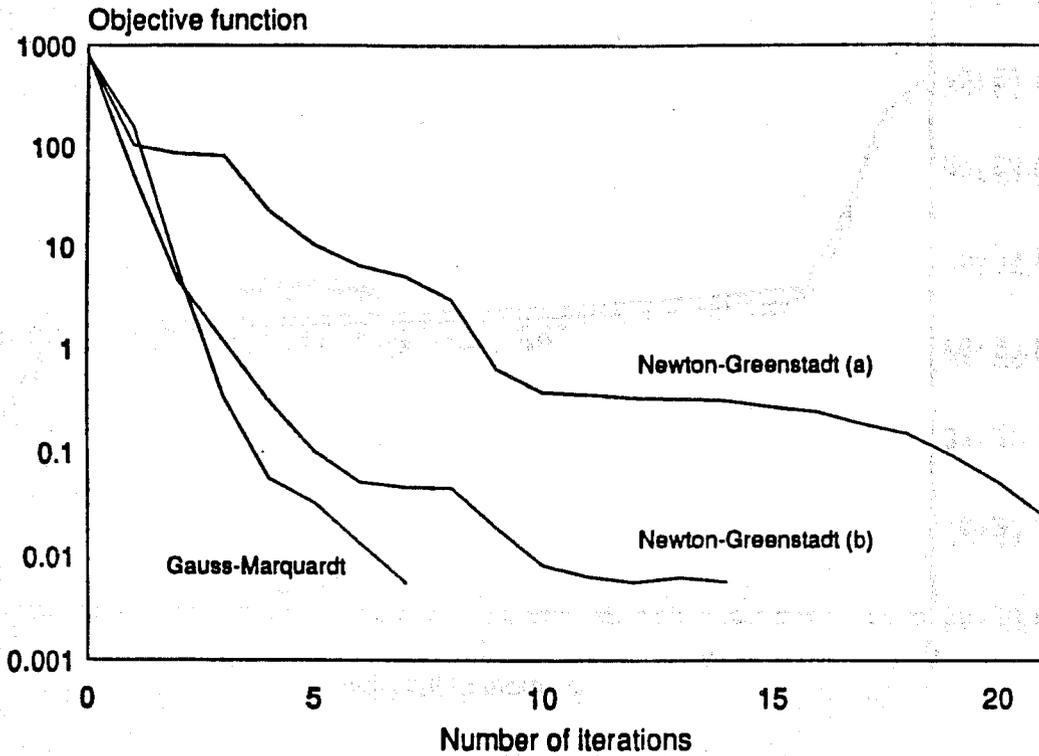


Figure 2.15: Double-porosity convergence

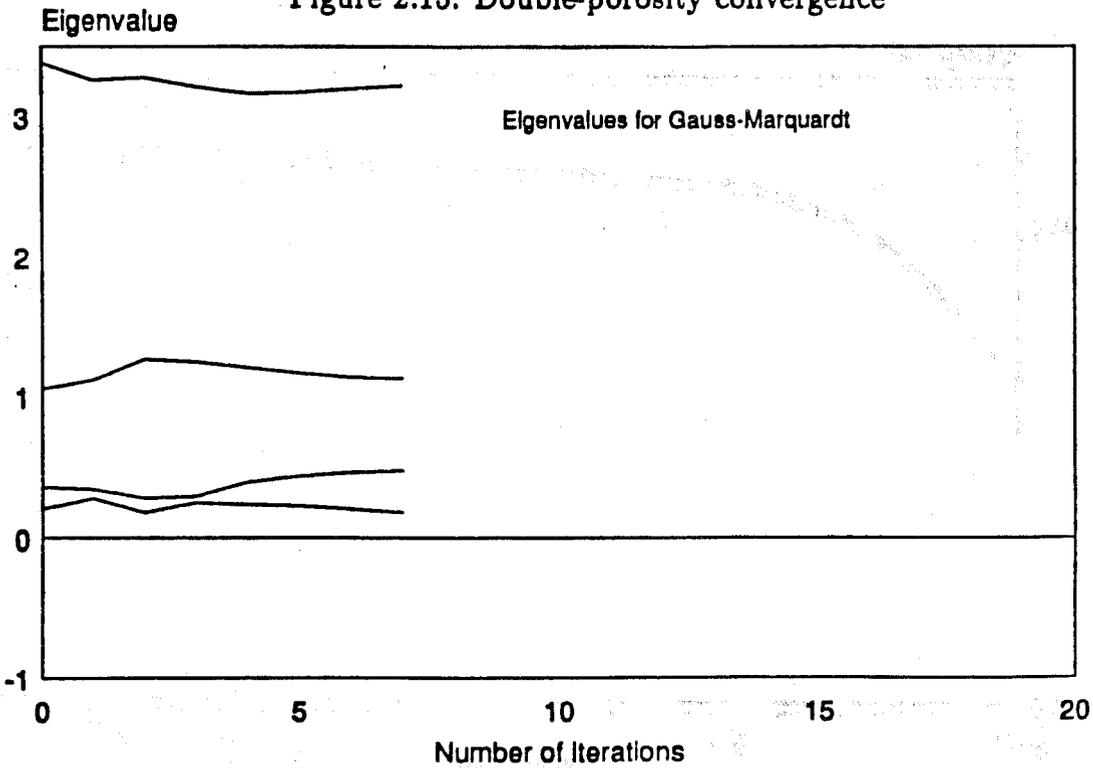


Figure 2.16: Eigenvalues for Gauss-Marquardt

Figure 2.17 shows the convergence with the addition of the sixth unknown (ϕ_c). Once again both Newton methods are unaffected by the introduction of an additional ill-defined parameter, while Gauss-Marquardt is, as expected, greatly affected. Newton-Greenstadt (a) is intolerably slow as before, but there is relatively little change in its performance with the additional ill-defined parameter. Once again, the restriction of the step as in Newton-Greenstadt (b) provides a greater benefit. The eigenvalues shown in Fig. 2.18 confirm the presence of two very small eigenvalues in the Gauss-Marquardt method once again.

2.4.6. Storage and Skin, Case 5

One of the objectives of this study was to find algorithms that would work even if some parameters were included that had not affected the reservoir response. This was checked by again analyzing the storage and skin test with the same initial guesses as Case 1, but fitting it this time with the two-porosity model.

Figure 2.19 shows that both Newton and Gauss-Marquardt remain unaffected and converge in almost exactly the same manner as when fitting the data with the homogeneous reservoir model.

Figure 2.20 shows the eigenvalues for the Gauss-Marquardt method and notice that there is still only one small eigenvalue, hence the rapid convergence. For this test it is obvious that there is no problem in using the double porosity model even though no double porosity behavior is observed. One explanation could be that the homogeneous reservoir response is a subset of the double porosity response. Nanba and Horne (1988) looked at a more severe problem of missing data. They looked at a data range where the effects of wellbore storage was dominant. Some difficulty in convergence was encountered in trying to estimate the three standard parameters (k , s and C). On the other hand no difficulty was encountered when storage data was missing. It may be that if most of the parameters can be found with the available data then convergence is rapid.

2.5. SUMMARY

- Comparison of first and second order methods shows that despite the extra information available, second order methods in general tend not to do as well as first order methods. First order methods usually converge faster unless good initial guesses are available for the second order methods to begin with.
- The first order Gauss-Marquardt method is greatly affected by the introduction of ill-defined parameters while second order methods are less so. This effect is noted even in the Newton-Greenstadt(a) method which does not use measures to handle ill-defined parameters.
- We use a Greenstadt type modification where small and negative eigenvalues are replaced by large positive numbers to improve the performance of Newton's method. With this modification the method becomes robust enough to achieve performance close to Gauss-Marquardt while remaining unaffected by the introduction of ill-defined parameters.
- Experiments show that if there is only one small eigenvalue of the Hessian matrix the Gauss-Marquardt method works very well. With ill-defined parameters the method works poorly and this seems to be correlated to the presence of more than one small eigenvalue.

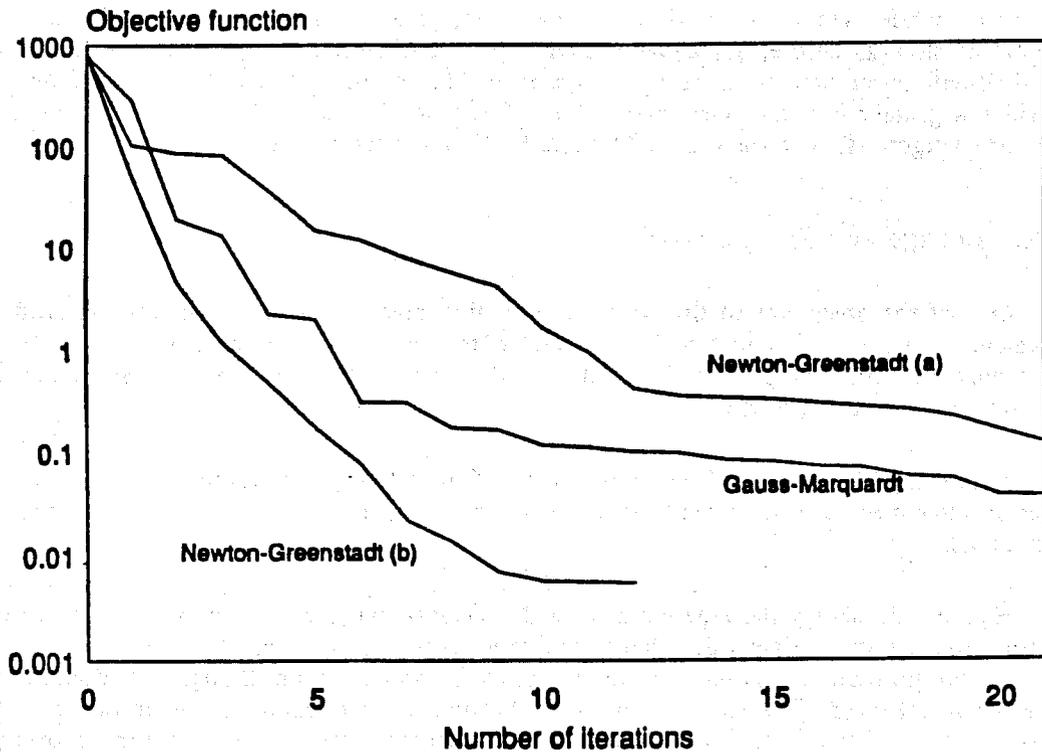


Figure 2.17. Convergence, double porosity test, 6 parameters.

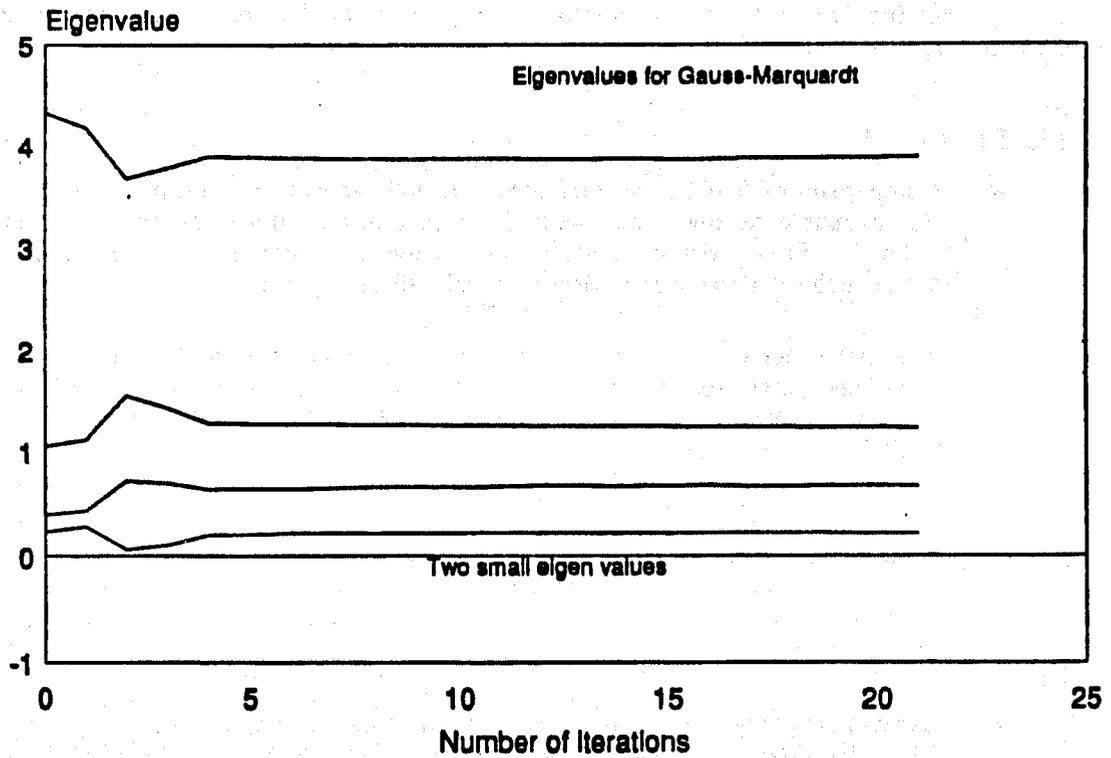


Figure 2.18. Double-porosity test eigenvalues, 6 parameters.

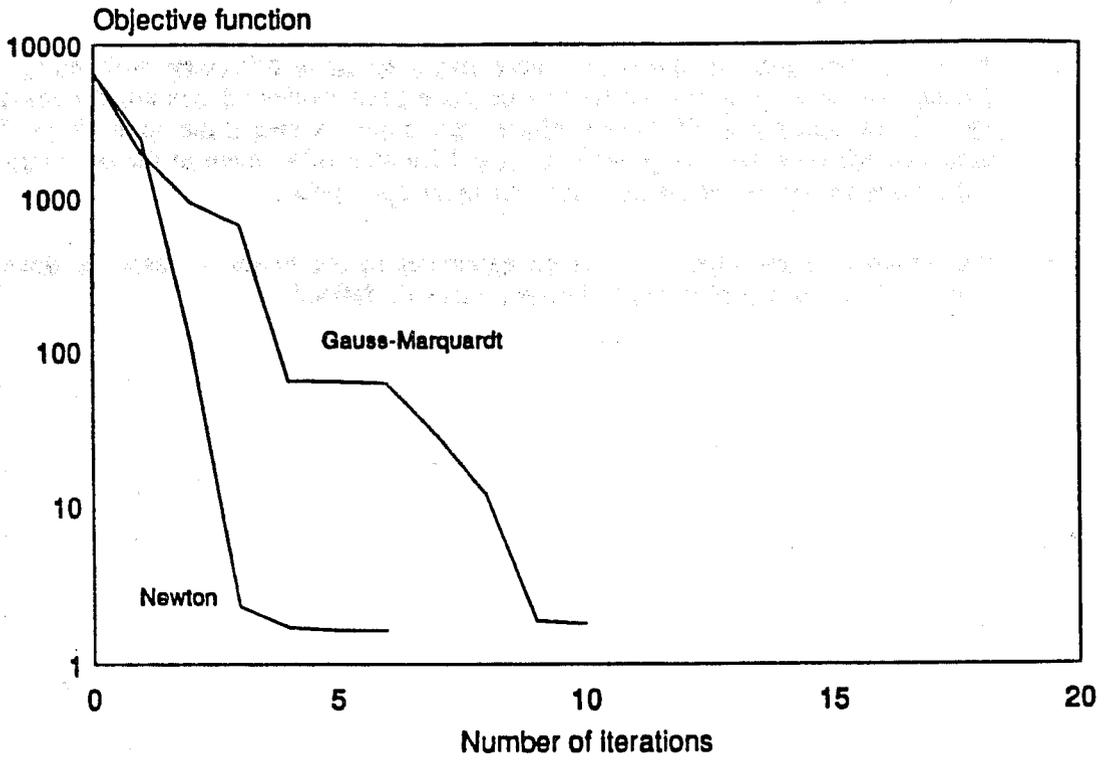


Figure 2.19. Fit with Double Porosity model

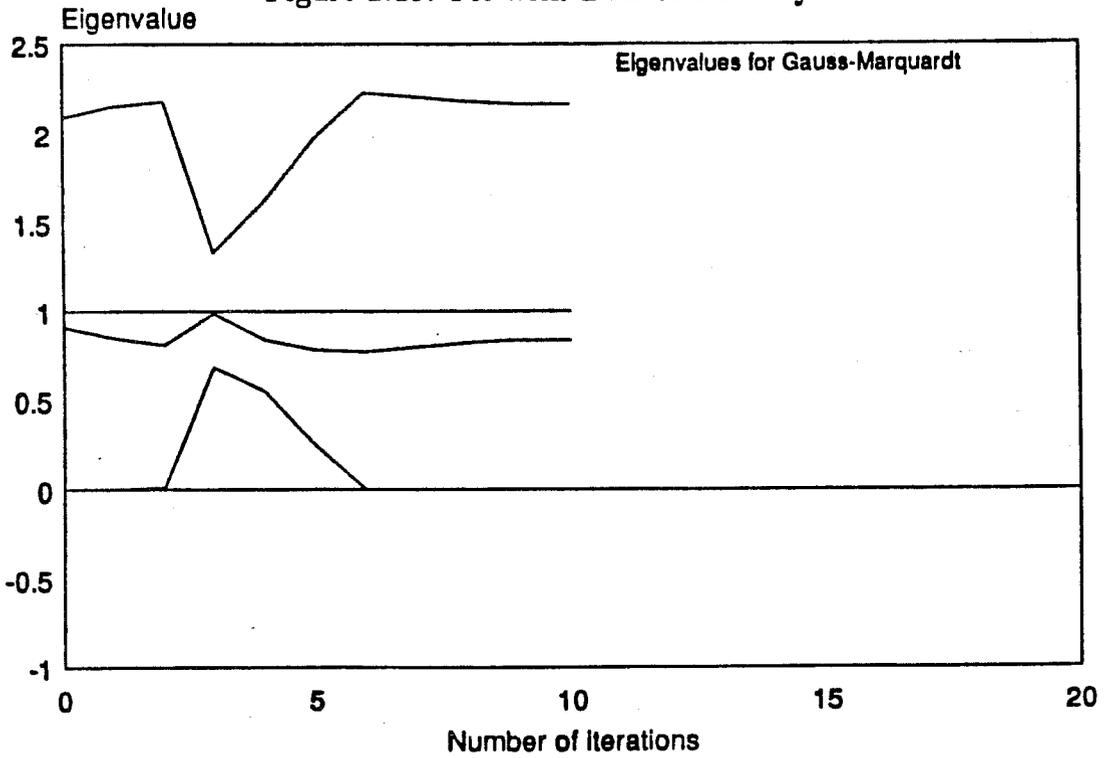


Figure 2.20. Eigenvalues for Gauss-Marquardt method

- Performance of the algorithms is unaffected by an increase in the number of parameters, provided no additional small eigenvalues are generated or ill-defined parameters introduced.
- Note that first-order methods are more likely to have difficulty with noisy data because the second derivative terms that have been neglected contain the residuals $(y - F)$ as multipliers. If these neglected terms are not small, the approximate Hessian will not necessarily approach the actual Hessian of F , even at the optimum. In such cases second order methods may be more appropriate.
- The absence of the effect of a given parameter in the reservoir response does not necessarily cause the problem to become more ill-defined.

3. PRODUCTION AND INJECTION SCHEDULE OPTIMIZATION

Optimization is closely related to automated well test analysis. When second order methods are used, both problems are mathematically identical. Optimization of production and injection schedules of EOR processes is thus a natural application of the work described in Chapter 2. Specifically, we looked at optimization of the cyclic steam injection process.

3.1. DESCRIPTION OF THE PROBLEM

Cyclic steam injection is the most widely used enhanced oil recovery process. The process is used on heavy oils that are too viscous to flow at normal reservoir temperatures. A cycle begins with the injection of steam for a few days to heat the reservoir. This is followed by a soak period when the well is shut in to allow the steam to condense. The well is then put on pump and produced until some economic limit is reached, at which point a fresh cycle begins.

Prats (1977) points out that because this process is relatively easy and inexpensive to implement, it is customarily tested directly in the field. Optimization can be done directly in the field especially if large amounts of data from several wells are available. The field data is sorted and the performance plotted with respect to operating conditions like reservoir characteristics, cumulative steam injection, soak time, etc. Typical responses are then assigned for each operating condition and these can be used to optimize the operating variables.

It is simple to optimize one operating variable by using historical performance curves or a mathematical model. However things become difficult if several variables are to be optimized, especially if their effects on the response are not independent of each other. To optimize two variables the single performance curve for one variable becomes a family of curves, each curve corresponding to a value for the second variable. This family of curves (or a part of it) has to be generated by numerical experimentation or from field data. With several variables the problem can become too large to handle this way. However it can be solved with much less effort by numerically optimizing a mathematical model.

3.2. MATHEMATICAL MODEL

Mathematical models for this process range from numerical simulators down to simplified models. Numerical simulators offer the most detail and accuracy. However they are time-consuming and also require a large amount of input data, much of which may not be known.

Several simplified models have been developed over the years (see Prats, 1977, for a summary). All of the models use an approximation to describe the extent of the steam zone. The *frontal displacement* model assumes a cylindrical front centered on the well. The *steam overlay* model assumes a uniform thickness of steam overlaying the oil.

Gontijo and Aziz (1984) proposed a model that more closely approximates the steam zone by assuming it to be of a conical shape. Since steam rises to the top of the reservoir this is more realistic than the frontal advance model, which does not allow override. At the same time, gravity drainage of oil keeps the oil column small near the well. Here again the conical model is more realistic than modeling the process with uniform thicknesses of steam and oil.

The model assumes that a thin layer of heated oil on the surface of the cone flows by a combination of gravity forces and pressure drop to the well. Several known results and correlations are used in the model, some of which are mentioned below, the reader is referred to Gontijo and Aziz (1984) for a more detailed description.

The model first obtains the average steam zone thickness h_{st} using an approximation (Van Lookeren, 1977) based on the total reservoir thickness, quantity of steam injected and the densities of oil and steam. The radius R_h of the heated zone is next calculated based on the volume of reservoir rock and fluids heated to steam temperature.

The production period is then modeled as a series of small time steps where flow rates and heat losses are computed. Each time step requires the estimation of the average temperature using the Boberg and Lantz (1966) equation:

$$T_{avg} = T_R + (T_s - T_R)[f_{HD}f_{VD}(1 - f_{PD}) - f_{PD}] \quad (3.1)$$

where f_{HD} and f_{VD} are dimensionless terms that account for heat losses from horizontal and vertical faces of the heated zone respectively, and f_{PD} accounts for heat lost with the produced fluids. This equation has been derived for the frontal advance model so its use in the conical model is an approximation. The dimensionless terms are calculated using the expressions:

$$f_{HD} = \frac{1}{1 + 5t_{DH}}$$

$$f_{VD} = \frac{1}{\sqrt{1 + 5t_{DV}}}$$

$$f_{PD} = \frac{1}{2Q_{max}} \int_0^t Q_p dt$$

where

$$t_{DH} = \frac{\alpha(t - t_{inj})}{R_h^2}$$

$$t_{DV} = \frac{4\alpha(t - t_{inj})}{h_t^2}$$

$$Q_{max} = H_{inj} + H_{last} - \pi R_h^2 K_R (T_s - T_R) \sqrt{\frac{t_{soak}}{\pi\alpha}}$$

$$Q_p = (q_o M_o + q_w M_w)(T_{avg} - T_R)$$

Q_p represents heat lost with produced fluids, while Q_{max} represents the total heat in the reservoir at the start of the production period. The model assumes heat losses begin after the injection period.

Once average temperature is known from Eq. 3.1, average viscosity, v_{avg} , can be obtained from correlations or curve-fit data. Gontijo and Aziz (1984) derived the following expression for flow rate in the cone:

$$q_l = 2\pi R_x \sqrt{\frac{kk_{r1}\phi\alpha\Delta S_1\Delta\Phi}{m_1 v_{avg}[\ln(R_x/r_w) - 0.5]}} \quad (3.2)$$

where

$$R_x = \sqrt{R_h^2 + h_i^2}$$

and ΔS_1 is the change in saturation following exposure to steam. The parameter m_1 is adjusted by doing a history match to observed production data. Gravity forces along the slope of the cone are included in the potential drop $\Delta\phi$.

Each time step consists of estimating average temperature using Eq. 3.1 and then calculating flow rates using Eq. 3.2. Time steps repeat with progressively decreasing temperatures and rates until the time allocated for production is used up.

The model is highly simplified but it attempts to account for all the major factors that play a role in the process. It has been shown by Gontijo and Aziz (1984) to be able to adequately match a numerical simulator's output and a real field well's performance.

The major benefit of its simplicity is that the model uses very little computer time, so it is ideally suited for optimization studies in this respect. We used the model to see how the same field well should have been optimally operated.

3.3. OPTIMIZATION

Among early optimization studies, Rivero and Heintz (1975) tried different financial objectives (present worth and cumulative daily profit) to optimize the time for beginning a new cycle. For the first objective they used an assumed rate response (a straight-line decline of peak oil rate with cumulative production) and attempted to maximize net present worth. For the second method they used routine expense and income records to determine when to begin a new cycle. Bentsen and Donahue (1969) used dynamic programming techniques to determine the allocation of a steam generator among wells.

Linear and nonlinear programming techniques have been used to optimize various other processes in the industry particularly in downstream processing plants. Lasdon *et al.* (1986) point out that the petroleum production area has seen few successful applications of optimization methods. They used nonlinear optimization techniques, specifically the BFGS Quasi-Newton (Broyden, 1970) method, with a single phase two-dimensional reservoir simulator to optimize a dry gas reservoir. They also suggested parallel processing to cut down on the large expense involved. See and Home (1983) applied linear programming techniques to a black oil reservoir simulator and proposed a method to reduce the number of simulation experiments needed.

3.3.1. Newton's Method with Finite Differences

In optimization the quadratic model no longer has the structure that allowed use of the Gauss method in the least-squares case. Newton's method is applicable of course, albeit with some modifications to ensure positive-definiteness. The quadratic model is once again

$$F^* = F_1 + g_1^T (x - x_1) + \frac{1}{2} (x - x_1)^T H_1 (x - x_1) \quad (3.3)$$

where \mathbf{x} is the vector of n nonlinear parameters, \mathbf{g} is the gradient vector and \mathbf{H} is the Hessian matrix. And the iterative scheme is:

$$\mathbf{H}_i \mathbf{p}_i = -\mathbf{g}_i \quad (3.4)$$

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \rho \mathbf{p}_i \quad (3.5)$$

Unlike in the least-squares case, the objective function F is now directly generated by the mathematical model of the process. Typically this model will be much too complicated to obtain analytical expressions for the elements of \mathbf{g} and \mathbf{H} so they have to be obtained by finite-difference. If done the obvious way, this can be very expensive. Instead we use the following relations. Let \mathbf{e}_i denote the i th unit vector, then the elements of gradient are obtained by

$$g_i = \frac{F(\mathbf{x} + h\mathbf{e}_i) - F(\mathbf{x} - h\mathbf{e}_i)}{2h}$$

the diagonal elements of the Hessian by

$$H_{ii} = \frac{F(\mathbf{x} + h\mathbf{e}_i) - 2F(\mathbf{x}) + F(\mathbf{x} - h\mathbf{e}_i)}{h^2}$$

and the off-diagonal elements of the Hessian by

$$H_{ij} = \frac{F(\mathbf{x} + h\mathbf{e}_i + h\mathbf{e}_j) - F(\mathbf{x} + h\mathbf{e}_i) - F(\mathbf{x} + h\mathbf{e}_j) + F(\mathbf{x})}{h^2}$$

The first two expressions are obtained by subtraction and addition of Taylor series expansions for $F(\mathbf{x} + h\mathbf{e}_i)$ and $F(\mathbf{x} - h\mathbf{e}_i)$ and are $O(h^2)$. Since $F(\mathbf{x})$ is always evaluated, the diagonal of the Hessian comes for free with the gradient.

The last expression is obtained by substituting the expression for the gradient in the Taylor series expansion for $F(\mathbf{x} + h\mathbf{e}_i)$ and is $O(h)$. It is efficient, for we obtain two off-diagonal Hessian elements (since the Hessian is symmetric) for each extra function evaluation.

The selection of h is somewhat difficult, for if it is too small cancellation errors may make the derivatives meaningless. If it is too large then the accuracy of the derivative suffers. Tests show that a fairly large h is preferable, we use $h_i = 0.1x_i$.

The Newton-Greenstadt (b) method can be used to modify \mathbf{H} once again. One difference now is that optimization frequently requires consideration of constraints when determining the optimum. We can still use penalty functions but there is a problem when the optimum lies directly on a constraint. Ideally that variable should be removed from the active set (Gill, Murray and Wright, 1983) so as not to affect the search process any more. With penalty functions we simply keep it from ever reaching the constraint.

By appropriately choosing the variables we can avoid this problem most of the time. In instances where the problem does occur the method simply keeps the variable close to the constraint and as will be shown later, we can still reach the optimum.

3.3.2. Quasi-Newton Methods for Optimization

Even though it is possible to evaluate the gradient and Hessian in a relatively efficient manner for Newton's method, a Quasi-Newton method might be able to do the same optimization with fewer function evaluations. The Broyden, Fletcher, Goldfarb, Shanno (BFGS) method [see, for example, Broyden (1970)], is particularly suitable for optimization. Essentially starting with \mathbf{I} , the Hessian is built up as the iterations proceed. The Hessian matrices satisfy the Quasi-Newton condition

$$\mathbf{H}_{k+1}(\mathbf{g}_{k+1} - \mathbf{g}_k) = \mathbf{x}_{k+1} - \mathbf{x}_k \quad (3.6)$$

This equation forces \mathbf{H}_{k+1} to exactly match the gradient of the function in the direction $\mathbf{x}_{k+1} - \mathbf{x}_k$, and so behave in a manner similar to Newton's method in this direction. Using the notation

$$\mathbf{y}_k \equiv \mathbf{g}_{k+1} - \mathbf{g}_k$$

$$\mathbf{s}_k \equiv \mathbf{x}_{k+1} - \mathbf{x}_k$$

the BFGS update may be written as

$$\mathbf{H}_{k+1} = \mathbf{H}_k + \frac{\mathbf{y}_k \mathbf{y}_k^T}{\mathbf{y}_k^T \mathbf{s}_k} - \frac{\mathbf{H}_k \mathbf{s}_k \mathbf{s}_k^T \mathbf{H}_k}{\mathbf{s}_k^T \mathbf{H}_k \mathbf{s}_k} \quad (3.7)$$

This update maintains symmetry and positive definiteness of the Hessian and also saves on function evaluations, so it is an attractive method. Furthermore its use has been well developed over the years and ready-made subroutines are available for constrained optimization using the BFGS method. We proceeded to optimize the process itself and at the same time compared the Newton method with the Quasi-Newton method using BCONG, a ready-made routine from IMSL (1987).

3.4. OPTIMIZATION EXPERIMENTS WITH THE MODEL

We optimized three major variables that are subject to operator control.

- First, the quantity of steam injected was optimized. We assumed that the generator capacity is limited to the rate actually used in the field, so injection time is used to vary steam quantity. Note that this automatically puts a limit on excessive injection volumes, for otherwise no time will be left for actual production.
- Second, the soak time was optimized, even though no oil is produced during this period while heat losses continue. The soak time is therefore theoretically unattractive but since such *a priori* knowledge may not always be available it was also included in the optimization, to allow the optimizer to show us this fact.
- Third, the producing time was optimized, although it was not treated as a separate variable, instead it was set to be the time left over in the cycle after steam injection and soak. So the task for the optimizer was to apportion a given total cycle time between injection, soak and production times.

We assumed that the steam is generated by burning some of the produced crude oil. To convert volume burned into heat supplied at the reservoir, a factor of 2.86 million BTU/bbl was used (5.72 million BTU/bbl at 50% efficiency).

Table 3.1 shows the effect of optimizing the net production, i.e. gross production less volume burned, for the first two cycles of the field test. These numbers are compared to the *base case*, which represents the steaming cycle actually used in the field. As expected the optimum soak times are close to zero (because of the penalty functions they cannot reach exactly zero). In practice some small soak time may be required to prevent backflow of steam or for running in the pump and preparing the well for production. Since soak time results in heat losses with little productive benefit, the time should clearly be minimized, perhaps by using some mechanical arrangement to prevent backflow.

Note also the greatly increased steam injection times. The optimal times are more than twice the times actually used in the field. The model obviously makes a strong case for high steam injection volumes, once a large amount of heat is input into the reservoir, high production rates can be sustained for a long period during production. This is similar to the prevailing practice in steam injection in Venezuela.

Looking at the nonlinear methods we see that the Newton method requires more iterations than the QN method. One of the reasons for this is that the soak times are at the constraint values and this is not handled well with penalty functions. Other reasons could be the efficacy of the line search and convergence criteria. Since the two programs are different in all these respects we instead can compare the number of function evaluations per iteration to see how efficient the methods are, it is clear that the QN method is much more efficient on this problem.

3.4.1. Optimizing a Linear Combination of Objectives

Notice in Table 3.1 that while optimizing net production the optimizer has done so at the expense of net production per barrel burned, three times as much steam is needed to double the net production. It may be desirable instead to obtain the best combination of these two economic objectives. This can be done by making the objective function a linear combination of the two primary objectives, each weighted according to the operator's priorities.

Table 3.2 shows the outcome of optimizing such a combination. The soak times are held at one day in view of the model's strong preference for zero soak times. So the remaining two operating variables are the injection times in first and second cycles.

Observe that in every case the net production obtained is greater than the net production for the base case. In Cases C and D the net/burned ratio is also greater than the base case value of 10.8 while net production is at least 55% higher. So if we assume that the model represents what will happen in reality, then it is clear that much can be done to improve the operating efficiency of the process both in terms of production rate and rate per barrel burned.

Notice also that there is some nonuniqueness in the optimum. For example in Case B, the optimum net production is only 25 barrels apart between the two methods but the gross and volume burned are significantly different.

This table shows that maximizing a combination of objectives could be more efficient than optimizing either one by itself. The most notable example is Case D, which shows very

Table 3.1. Optimizing net production (4 parameters)

		Base Case	Newton	QN
Cycle 1	Injection time	6	30.8	28.8
	Soak time	5	0.167	0.167
	Production time	55	35	37
Cycle 2	Injection time	9	14.8	18.4
	Soak time	2	0.67	0.67
	Production time	146	141.5	138.2
Totals	Gross Production	13228	27627	27783
	Net Production	12062	24280	24267
	Volume Burned	1166	3347	3516
	Net Prod/bbl brn	10.34	7.25	6.90
	Num. iterations		17	11
	Num. func. evals		291	119

Table 3.2. Optimizing a linear combination of objectives

	Inj. time 1	Inj. time 2	Gross Prodn.	Net Prodn.	Vol Burned	Net/ Burned	Obj. Func	Its	Func Evals
Field simulation									
	6	9	13228	12062	1166	10.34			
Case A: Unoptimized (1 day soak time)									
	6	9	13759	12593	1166	10.8			
Case B: $\mathcal{F} = \text{Net oil production}$									
N	28.7	16.0	27190	23884	3306	7.223	23884	7	61
QN	28.7	17.8	27313	23859	3454	6.907	23859	6	45
Case C: $\mathcal{F} = 0.8 * \text{Net} + 1000 * \text{Net/Burned}$									
N	22.94	0.3	21481	19889	1592	12.5	28411	22	182
QN	22.9	0.3	21445	19858	1587	12.51	28396	5	32
Case D: $\mathcal{F} = 0.7 * \text{Net} + 1500 * \text{Net/Burned}$									
N	16.5	0.3	17771	16618	1153	14.42	33263	7	60
QN	16.53	0.3	17793	16639	1154	14.41	33262	4	28

good production with less total volume burned than in the base case. In both Cases C and D it is apparent that the optimizer would like to avoid the second injection period. One reason for this preference is that the temperature of the injected steam in the first cycle is 30 degrees higher than in the second cycle, although the injection rate is lower. So the optimizer has judged it to be more worthwhile to spend time on the first injection period only.

Once again, as far as the nonlinear methods are concerned, the Newton method does not seem to do as well as Quasi-Newton. Function evaluations are once again more frequent in the Newton method. Note that by removing soak time from the problem the convergence is much improved, Case B is now fully unconstrained and requires 6-7 iterations compared to 11/17 in Table 3.1.

3.4.2. Effect of Steam Properties

The process is sensitive to parameters like formation thickness and pattern area which may be beyond the operator's control. But there are other parameters that can be controlled to some extent, such as steam temperature and quality. Steam temperature is solely determined by the injection pressure. However it is possible, for example, to inject at high rates so that the pressure (and thus temperature) goes up. Table 3.3 shows the result of increasing temperature in both cycles by 100 degrees Fahrenheit. Case A uses the unoptimized production schedule with the higher temperatures, and shows that performance has not improved by much over the field results. In contrast the production rates are now much higher when optimization is performed. For example, in Case B the net production is 70% higher than previously achieved. Even Cases C and D show greatly increased production although we are trying to maximize efficiency in these cases.

Table 3.4 shows the effect of keeping steam temperature unchanged and raising the quality by 0.15 in both cycles. In marked contrast to Table 3.3, the performance does not improve very much with this change.

By increasing steam quality we increase the total heat input to the reservoir so average temperature will decline less rapidly. Production rates will therefore also decline less rapidly. With a higher steam temperature, heat input is increased too, but more significantly the average temperature from Eq. 3.1 is always higher (assuming all else is unchanged), so the viscosity stays low and daily production rates are higher. These optimized results show that this situation allows for greater improvements than a simple increase in heat input.

3.4.3. Effect of Cycle Parameters

The previous results have indicated a strong tendency to favor the first cycle for its higher steam injection temperature. Table 3.5 shows what would happen if all steam properties were identical in both cycles. For net production there is now no preferential treatment for Cycle 1. However, once we consider efficiency, the tendency to favor Cycle 1 is again apparent.

Table 3.3. Optimizing with higher steam temperature

	Inj. time 1	Inj. time 2	Gross Prodn.	Net Prodn.	Vol Burned	Net/ Burned	Obj. Func	Its	Func Evals
Field simulation									
	6	9	13228	12062	1166	10.34			
Case A: Unoptimized									
	6	9	17986	16743	1243	13.47			
Case B: $\mathcal{F} = \text{Net oil production}$									
N	27.6	37	47083	41763	5319	7.85	41763	6	54
QN	27	37.8	47213	41856	5357	7.81	41856	7	52
Case C: $\mathcal{F} = 0.8 * \text{Net} + 1500 * \text{Net/Burned}$									
N	26	17.16	41674	38257	3417	11.2	47406	10	85
QN	26.5	15	40839	37583	3255	11.54	47376	6	47
Case D: $\mathcal{F} = 0.8 * \text{Net} + 3000 * \text{Net/Burned}$									
N	21.96	0.31	28004	26399	1605	16.45	71753	8	66
QN	21.95	0.3	27986	26383	1603	16.45	71667	6	39

Table 3.4. Optimizing with higher steam quality

	Inj. time 1	Inj. time 2	Gross Prod.	Net Prod.	Vol Burned	Net/ Burned	Obj. Func	Its	Func Evals
Field simulation									
	6	9	13228	12062	1166	10.34			
Case A: Unoptimized									
	6	9	16065	14704	1361	10.80			
Case B: $\mathcal{F} = \text{Net oil production}$									
N	28.5	12	29924	26500	3424	7.74	26500	13	106
QN	26.9	17.3	30386	26561	3825	6.94	26561	5	39
Case C: $\mathcal{F} = 0.8 * \text{Net} + 1000 * \text{Net/Burned}$									
N	23.7	0.3	25119	23223	1895	12.25	30828	5	44
QN	23.4	0.3	24946	23074	1872	12.33	30789	6	40
Case D: $\mathcal{F} = 0.7 * \text{Net} + 1500 * \text{Net/Burned}$									
N	16	0.3	20268	18980	1288	14.73	35381	7	61
QN	16.8	0.3	20882	19532	1350	14.47	35377	6	39

Table 3.5. Optimizing with balanced steam properties

	Inj. time 1	Inj. time 2	Gross Prodn.	Net Prodn.	Vol Burned	Net/ Burned	Obj. Func	Its	Func Evals
Field simulation									
	6	9	13228	12062	1166	10.34			
Case A: Unoptimized									
	6	9	14328	13303	1024	12.98			
Case B: $\mathcal{F} = \text{Net oil production}$									
N	29.5	28	32557	28634	3923	7.3	28634	7	62
QN	30.4	30	32692	28579	4113	6.9	28579	3	28
Case C: $\mathcal{F} = 0.8 * \text{Net} + 1000 * \text{Net/Burned}$									
N	27.7	12	29976	27264	2712	10.05	31861	5	41
QN	28.9	9.53	29595	26970	2625	10.27	31846	5	41
Case D: $\mathcal{F} = 0.8 * \text{Net} + 1500 * \text{Net/Burned}$									
N	20.8	0.31	22332	20891	1441	14.5	38463	8	63
QN	21.9	0.3	23558	21893	1575	14	38514	5	35

3.5. SUMMARY

The tests with a simplified model reveal that significant improvements in efficiency may be possible by using optimal operating policies. We have explored the effect of some of the operating conditions on the optimal policies. The specific numbers obtained are particular to the field test simulated but we can make some general observations.

- Soak time should be minimized or eliminated if possible.
- Significantly higher steam injection volumes than current practice may be beneficial. This allows subsequent production rates to be sustained for a longer period.
- While attempts to improve steam quality are desirable to reduce heat lost outside the reservoir, the effect of reservoir steam quality does not seem to be as significant as that of the steam temperature. With a higher steam temperature it may be possible to make very large improvements in production rate using optimal strategies.
- Optimizing a combination of objectives can lead to a more efficient process than optimizing just one objective by itself.
- Because of the large number of process variables that are needed to simulate this process, it will be difficult to give exact guidelines unless the correlations and the model are specifically tuned for the reservoir under consideration.
- Furthermore, because the model is simplified, it cannot simulate all the intricate details of the process, so recommendations from the model will need verification in the field followed by additional tuning.
- Because two different routines were used in the study, we cannot compare absolute performance of the two nonlinear methods but it is clear that the Quasi-Newton method can provide savings in function evaluations and at same time converge quite rapidly.

The simplified Gontijo and Aziz model used here is adequate for exploratory studies. Ideally we would like to use a model that can more rigorously model all the phenomena. This would have to be a numerical simulator, but the principal difficulty is that each optimization may take 50-100 full simulation runs. This would be very expensive. One way of addressing this problem would be to use parallel computers to either do the function evaluations in parallel, or more generally to run each individual simulation in a parallel manner. Quasi-Newton methods also hold promise for simulation. So a detailed study of parallel simulation and of Quasi-Newton methods was performed next. This is covered in subsequent chapters.

4. QUASI-NEWTON METHODS IN RESERVOIR SIMULATION

Quasi-Newton methods are applicable to both optimization and nonlinear equation solving. In optimization the usefulness of the method has been seen in the previous chapter. However the method's utility in equation solving is not so clear. The form of the Quasi-Newton method also seems attractive for parallel processing since one can directly solve the nonlinear problem of fully implicit reservoir simulation using this method, but whether this results in a more efficient method is again not clear. These questions are explored in this and subsequent chapters.

4.1. PREVIOUS WORK

Quasi-Newton methods have been used successfully on nonlinear systems of equations in the finite-element and ordinary differential equation fields to avoid some of the costs associated with Newton's method and to provide better convergence than the modified-Newton method. In the modified-Newton method the Jacobian is formed and factored once into the lower triangular matrix L and the upper triangular matrix U . Subsequent iterations only involve forward elimination with L followed by back substitution with U and so are cheaper than a Newton iteration.

Mattheis and Strang (1979) explored the use of Quasi-Newton methods on nonlinear finite-element equations and showed how a Quasi-Newton method could succeed where the modified-Newton method failed. Brown *et al.* (1985) experimented with Quasi-Newton methods on stiff systems of ordinary differential equations and showed that for some problems a Quasi-Newton method could do better than the modified-Newton method. Nghiem (1983) proposed a Quasi-Newton method which he named the QNSS and reported success with it in handling the pressure equation in a compositional simulator. Steihaug (1981) worked out the convergence of inexact Quasi-newton methods. In addition, work on new Quasi-Newton methods continues in the applied mathematics field, especially for sparse systems (for example see Kelley and Sachs (1987), and Tewarson and Zhang (1987)). Many of the practical details about Quasi-Newton methods are provided in Dennis and Schnabel (1983).

This part of the study was aimed at exploring whether one of the most popular Quasi-Newton methods could be used to good effect in reservoir simulation problems.

4.2. RESERVOIR SIMULATION

Black oil reservoir simulation requires the solution of the following set of coupled multi-phase fluid flow equations for oil, water and gas (for more details see Aziz and Settari, 1979):

$$\nabla[\lambda_o(\nabla p_o - \gamma_o \nabla z)] = \frac{\partial}{\partial t} \left[\frac{\phi(1 - S_w - S_g)}{B_o} \right] + q_o \quad (4.1)$$

$$\nabla[\lambda_w(\nabla p_w - \gamma_w \nabla z)] = \frac{\partial}{\partial t} \left[\frac{\phi S_w}{B_w} \right] + q_w \quad (4.2)$$

$$\nabla[R_s \lambda_o(\nabla p_o - \gamma \nabla z) + \lambda_g(\nabla p_g - \gamma_g \nabla z)] = \frac{\partial}{\partial t} \left[\phi \left(\frac{R_s(1 - S_w - S_g)}{B_o} + \frac{S_g}{B_g} \right) \right] + R_s q_o + q_g \quad (4.3)$$

These are the result of substituting Darcy's law into the mass conservation equations for oil, water and gas respectively, over a unit volume. The three phase pressures p_o , p_w and p_g are replaced by a single unknown, p_o , using the following relations:

$$P_{cow} = p_o - p_w \quad (4.4)$$

$$P_{cog} = p_o - p_g$$

where P_{cow} and P_{cog} are obtained by table lookup. The mobilities, λ_l , are defined as

$$\lambda_l = \frac{k k_{r_l}}{\mu_l B_l} \quad l = o, w, g \quad (5)$$

where relative permeability k_{r_l} is a function of saturation, $k_{r_l} = k_{r_l}(S_l)$. Viscosity μ_l , formation volume factor B_l and density γ_l are all functions of phase pressure p_l . All these properties are found by table lookup. The absolute permeability k and porosity ϕ are assumed to be known.

When free gas is present, the solution gas oil ratio $R_s = R_s(p_o)$ is found by table lookup, so the three unknowns are (p_o, S_w, S_g) . When no free gas is present $S_g = 0$, and the three unknowns are (p_o, S_w, R_s) . This change is handled by the process of variable substitution.

The fluid flow equations are discretized on a block-centered grid using the fully implicit backward-in-time centered-in-space approach. The result of applying this finite-difference scheme to the oil or water equation can be written as (in one dimension)

$$[\Delta T_l \Delta(p_l - \gamma_l z)]_i^{n+1} = \frac{V}{\Delta t} \Delta_t \left[\frac{\phi S_l}{B_l} \right] + Q_l, \quad = o, w \quad (4.6)$$

where V represents volume of the grid block and Q_l the block injection/production rate.

The flow terms on left hand side of Eq. 4.6 can be expanded as

$$\begin{aligned} \Delta T \Delta(p - \gamma z) &\equiv \Delta T \Delta p - \Delta T \gamma \Delta z \\ &\equiv T_{i+\frac{1}{2}} [p_{i+1} - p_i - \gamma_{i+\frac{1}{2}} (z_{i+1} - z_i)] \\ &\quad - T_{i-\frac{1}{2}} [p_i - p_{i-1} - \gamma_{i-\frac{1}{2}} (z_i - z_{i-1})] \end{aligned} \quad (4.7)$$

where interblock transmissibility $T_{i+1/2} = \lambda_{i+1/2} (A_{i+1/2} / \Delta x_{i+1/2})$. $A_{i+1/2}$ is the cross-sectional area available for flow between block i and block $i + 1$, and $\Delta x_{i+1/2}$ is the distance between the centers of these grid blocks.

The accumulation term on the right-hand side of Eq. 4.6 can be expanded as

$$\frac{V}{\Delta t} \Delta_t \left[\frac{\phi S_l}{B_l} \right] = \frac{V}{\Delta t} \left[\left[\frac{\phi S_l}{B_l} \right]^{n+1} - \left[\frac{\phi S_l}{B_l} \right]^n \right] \quad (4.8)$$

The superscript n indicates values evaluated at the current time t and $n + 1$ indicates values at time $t + \Delta t$. All terms on the left hand side of Eq. 4.6 are evaluated at time level $n + 1$.

Physical properties at the boundary, *i.e.* at $i + 1/2$, are set to the average of the values in the two adjacent blocks, except for relative permeability where upstream weighting is used, *i.e.*

$$(k_r)_{i+1/2} \begin{cases} \text{if } (p_i - \gamma_{i+1/2} z_i) > (p_{i+1} - \gamma_{i+1/2} z_{i+1}) \\ \text{otherwise} \end{cases}$$

Equation 4.6 is expanded in a similar manner for the other two dimensions. Similarly, the three-dimensional equations for water and gas are also discretized.

By moving the right-hand side terms of Eq. 4.6 to the left, we can write the equation in residual form. In the one dimensional case this gives:

$$\begin{aligned} T_{i+1/2} [p_{i+1} - p_i - \gamma_{i+1/2} (z_{i+1} - z_i)] - T_{i-1/2} [p_i - p_{i-1} - \gamma_{i-1/2} (z_i - z_{i-1})] \\ - \frac{V}{\Delta t} \left[\left[\frac{\phi S_1}{B_1} \right] - \left[\frac{\phi S_1}{B_1} \right]^n \right] - Q_1 = 0 \end{aligned} \quad (4.9)$$

with all terms except the accumulation term with superscript n , evaluated at time level $n + 1$. This is the *fully implicit* formulation of the reservoir simulation equations and is unconditionally stable for any time step size. When written for all phases and grid blocks, these form a nonlinear system of equations:

$$f(x) = 0 \quad (4.10)$$

that must be solved for x , the vector of unknowns (p_o , S_w , R_s , or S_g) at each grid block.

The classical technique for solving this nonlinear system of equations, from an initial guess x_0 , is Newton's method, as shown in Fig. 4.1.

for $k = 0, 1, \dots$ until converged
 Form Jacobian matrix J_k at x_k
 Solve $J_k s_k = -f(x_k)$
 Add step, $x_{k+1} = x_k + s_k$

Figure 4.1. Newton's method for solving nonlinear systems of equations

Newton's method is quadratically convergent when started with a good initial guess. However it is expensive since the Jacobian and the solution of the matrix problem, $J_k s_k = -f(x_k)$, must be computed at each iteration.

Quasi-Newton methods can also be used to solve the nonlinear system of equations $f(x) = 0$ and attempt to reduce some of the expense associated with Newton's method. Such methods have a slower convergence rate than Newton's method but the cost of each iteration is lower, so in terms of total time they may be more efficient.

4.3. QUASI-NEWTON METHODS FOR SYSTEMS OF EQUATIONS

Quasi-Newton methods are based on building up secant information as the iterations proceed. Suppose at the k th iteration the method takes a step $s_k \equiv x_{k+1} - x_k$, causing a change $y_k \equiv f(x_{k+1}) - f(x_k)$ to the nonlinear system of equations. Then the next Jacobian approximation, A_{k+1} , will satisfy the secant passing through these two iterates if

$$A_{k+1} s_k = y_k \quad (4.11)$$

This is termed the Quasi-Newton equation or the Quasi-Newton condition and is the design criterion for such methods. Equation 4.11 has two known vectors, s_k and y_k , and an unknown matrix A_{k+1} . It is impossible to uniquely solve Eq. 4.11 for the unknown matrix except in the one-dimensional case, when one simply obtains a single number equal to the slope of the secant.

Since a large number of matrices satisfy Eq. 4.11, several different Quasi-Newton schemes have been proposed. All of these attempt to update the current Jacobian approximation A_k rather than obtain a new approximation A_{k+1} from scratch.

Two of the most popular Quasi-Newton schemes for nonlinear equations were tried on reservoir simulation problems during this study. These are the BFGS (Broyden, 1970) update and Broyden's (Broyden, 1965) first method. The BFGS update has certain desirable properties, such as maintaining the symmetry and positive-definiteness of the Jacobian, which makes it very effective in optimization. Early testing showed that Broyden's update works better for the nonsymmetric Jacobians that arise in reservoir simulation.

Two differences occur when using a Quasi-Newton method instead of Newton's method. First, the convergence rate is at most q -superlinear (*i.e.* faster than linear but possibly either much faster or only a little faster than linear).

The second difference is that s_k , the step taken at each iteration is now the solution to the equation:

$$A_k s_k = -f(x_k) \quad (4.12)$$

In other words, it is the solution to the approximate Jacobean that has been built up using secant information by updating A_{k-1} in some manner. In turn A_{k-1} has been obtained from A_{k-2} and so on down to A_0 .

Simply obtaining one matrix from another is, of course, not very interesting if the latest matrix has to be solved from scratch, since then Newton's method would perform better. Instead the updates are designed such that all solutions after the first will be less expensive.

4.4. BROYDEN'S UPDATE

Broyden's update may be applied either directly to the inverse of the approximate Jacobian matrix or to the factors of the approximate Jacobian matrix. The latter approach, suggested by Gill and Murray (1972), updates the QR factors of the Jacobian matrix, this approach is numerically very stable because it uses Householder transformations to triangularize the matrix. However the QR factorization is twice as expensive as LU factorization, so we do not consider this form of update further.

The inverse form of Broyden's update provides a convenient way to solve Eq. 4.12 on a parallel computer. It updates the inverse of the Jacobian, so that the solution of Eq. 4.12 could, in principle, be obtained by the matrix-vector product: $\mathbf{s}_k = -\mathbf{A}_k^{-1} \mathbf{f}(\mathbf{x}_k)$.

This update is given as:

$$\mathbf{A}_{k+1}^{-1} = \mathbf{A}_k^{-1} + \frac{(\mathbf{s}_k - \mathbf{A}_k^{-1} \mathbf{y}_k) \mathbf{s}_k^T \mathbf{A}_k^{-1}}{\mathbf{s}_k^T \mathbf{A}_k^{-1} \mathbf{y}_k} \quad (4.13)$$

This is a recurrence relation, so once \mathbf{A}_0^{-1} is obtained by some means, then using Eq. 4.13, it can be converted into \mathbf{A}_1^{-1} and so on without any further inversions. Since matrix-vector products are highly parallelizable, all iterations after the first will be performed very quickly.

The inverse matrices are never actually formed for several reasons. Obtaining the inverse of a matrix is costly and numerically undesirable. The inverse of a sparse matrix will usually be dense, requiring substantial storage. Finally, the matrix-vector product will be expensive on a dense matrix.

Instead we store either \mathbf{A}_0^{-1} or the factors of \mathbf{A}_0 and a series of update vectors, and this is how the updates are usually applied (Mattheis and Strang, 1979). The detailed steps for Broyden's update are provided later in this chapter.

4.4.1. Properties of Broyden's Update

The obvious property of Broyden's update is that it satisfies the Quasi-Newton equation. A second property is that all directions other than \mathbf{s}_k are unaffected by the update from \mathbf{A}_k to \mathbf{A}_{k+1} . The update modifies the approximate Jacobian by using only the information obtained in the last step and does not introduce any extraneous information. This second property implies that $\mathbf{A}_k \mathbf{x} = \mathbf{A}_{k+1} \mathbf{x}$, for all \mathbf{x} orthogonal to \mathbf{s}_k , i.e., $\mathbf{s}_k^T \mathbf{x} = 0$.

These two properties can be regarded as the design conditions for deriving Broyden's update. Since information is only obtained in one direction we start with a prototype of a *rank one* correction for the update, of the form:

$$\mathbf{A}_{k+1}^{-1} = (\mathbf{I} + c_k \mathbf{s}_k^T) \mathbf{A}_k^{-1} \quad (4.14)$$

This form satisfies the second property. The unknown c_k can be obtained by substituting this update into the Quasi-Newton equation as follows:

$$\begin{aligned} \mathbf{s}_k &= \mathbf{A}_{k+1}^{-1} \mathbf{y}_k \\ &= (\mathbf{I} + c_k \mathbf{s}_k^T) \mathbf{A}_k^{-1} \mathbf{y}_k \\ &= \mathbf{A}_k^{-1} \mathbf{y}_k + c_k \mathbf{s}_k^T \mathbf{A}_k^{-1} \mathbf{y}_k \end{aligned}$$

Therefore

$$c_k = \frac{\mathbf{s}_k - \mathbf{A}_k^{-1} \mathbf{y}_k}{\mathbf{s}_k^T \mathbf{A}_k^{-1} \mathbf{y}_k}$$

which when substituted into Eq. 4.14 gives Broyden's update.

The step vectors s_k are not guaranteed to be orthogonal; they could even be linearly dependent so, unlike the one-dimensional secant method, the precise order of local convergence of Broyden's method cannot be stated.

Dennis and Schnabel (1984) show that if the conditions for quadratic convergence of Newton's method are met and, if there exist positive constants ϵ , δ such that $\|x_0 - x_*\| \leq \epsilon$ and $\|A_0 - J(x_*)\| \leq \delta$, then the sequence x_k generated by Broyden's method is well defined and converges q -superlinearly to the solution x_* . Here $\|\cdot\|$ denotes the matrix or vector l_2 norm.

The proof requires

$$\|J(x_*)^{-1}\| \|A_0 - J(x_*)\| \leq 1/6$$

This condition is sufficient to prove q -superlinear convergence, which implies that the Quasi-Newton step approaches the Newton step in both direction and magnitude.

They point out that the *order* of the q -superlinear convergence cannot be stated unless one can also state the convergence rate of the limit:

$$\lim_{k \rightarrow \infty} \frac{\|(A_k - J(x_*))s_k\|}{\|s_k\|} = 0$$

An *a priori* estimate of this convergence rate will, in general, be difficult to obtain for a real problem. So the most that can be said is that if the initial approximate Jacobian is close to the Jacobian at the solution then one can expect a q -superlinear convergence rate (of an unspecified order).

They also show that if we just assume Lipschitz continuity of the Jacobian in the search space D :

$$\|J(x) - J(x_*)\| \leq \gamma \|x - x_*\|, \forall x \in D$$

then we can at least expect linear convergence. This serves as a lower bound on the convergence rate to be expected from Broyden's method.

The Lipschitz continuity condition will usually be met by the Jacobians in reservoir simulation, small discontinuities appear when wells start declining after flowing at a specified rate, larger discontinuities appear when phase changes occur but we can avoid trouble with such changes as described later.

With only linear convergence guaranteed and lacking any more definite convergence result one has to rely on numerical experiments to see if and when the method provides any advantages.

4.4.2. Implementing the Update Efficiently

To see how Broyden's update, Eq. 4.13 can be efficiently applied using update vectors consider the following. Recall Broyden's update:

$$A_{k+1}^{-1} = A_k^{-1} + \frac{(s_k - A_k^{-1}y_k)s_k^T A_k^{-1}}{s_k^T A_k^{-1} y_k}$$

where

$$y_k \equiv f_{k+1} - f_k$$

$$s_k \equiv x_{k+1} - x_k$$

$$= -A_k^{-1} f_k$$

This update can be written in the form:

$$A_{k+1}^{-1} = (I + c_k s_k^T) A_k^{-1}$$

where

$$s_k = -A_k^{-1} f_k$$

$$b_k \equiv A_k^{-1} f_{k+1}$$

$$c_k \equiv \frac{-b_k}{s_k^T (b_k + s_k)}$$

We do not want to actually construct A_{k+1}^{-1} so to see how the iterations proceed consider the first three iterations. At the first iteration we have:

$$s_0 = -A_0^{-1} f_0$$

which is the same as the Newton step if $A_0 = J_0$. At the second iteration we have:

$$b_0 = A_0^{-1} f_1$$

$$c_0 = \frac{-b_0}{s_0^T (b_0 + s_0)}$$

$$\begin{aligned} s_1 &= -A_1^{-1} f_1 \\ &= -(I + c_0 s_0^T) A_0^{-1} f_1 \\ &= -(I + c_0 s_0^T) b_0 \end{aligned}$$

At the third iteration we have:

$$\begin{aligned} b_1 &= A_1^{-1} f_2 \\ &= (I + c_0 s_0^T) A_0^{-1} f_2 \end{aligned}$$

$$c_1 = \frac{-b_1}{s_1^T (b_1 + s_1)}$$

$$\begin{aligned} s_2 &= -A_2^{-1} f_2 \\ &= -(I + c_1 s_1^T) A_1^{-1} f_2 \\ &= -(I + c_1 s_1^T) b_1 \end{aligned}$$

At the fourth iteration we have:

$$\begin{aligned}
 \mathbf{b}_2 &= \mathbf{A}_2^{-1} \mathbf{f}_3 \\
 &= (\mathbf{I} + \mathbf{c}_1 \mathbf{s}_1^T) \mathbf{A}_0^{-1} \mathbf{f}_3 \\
 \mathbf{c}_2 &= \frac{-\mathbf{b}_2}{\mathbf{s}_2^T (\mathbf{b}_2 + \mathbf{s}_2)} \\
 \mathbf{s}_3 &= -\mathbf{A}_3^{-1} \mathbf{f}_3 \\
 &= -(\mathbf{I} + \mathbf{c}_2 \mathbf{s}_2^T) \mathbf{A}_2^{-1} \mathbf{f}_3 \\
 &= -(\mathbf{I} + \mathbf{c}_2 \mathbf{s}_2^T) \mathbf{b}_2
 \end{aligned}$$

So at every iteration we obtain a tentative step, $\mathbf{d}_k \equiv \mathbf{A}_0^{-1} \mathbf{f}_k$, which then goes through some updates to finally obtain the step \mathbf{s}_k (computing \mathbf{b}_{k-1} and \mathbf{c}_{k-1} on the way). The numbering scheme is a little confusing since each iteration uses information from the previous iteration(s), but it is relatively easy to program the algorithm.

The updates can be applied with little cost after \mathbf{d}_k is obtained. Looking at part of the fourth iteration again:

$$\begin{aligned}
 \mathbf{b}_2 &= \mathbf{A}_2^{-1} \mathbf{f}_3 \\
 &= (\mathbf{I} + \mathbf{c}_1 \mathbf{s}_1^T) (\mathbf{I} + \mathbf{c}_0 \mathbf{s}_0^T) \mathbf{A}_0^{-1} \mathbf{f}_3 \\
 &= (\mathbf{I} + \mathbf{c}_1 \mathbf{s}_1^T) (\mathbf{I} + \mathbf{c}_0 \mathbf{s}_0^T) \mathbf{d}_3 \\
 &= (\mathbf{I} + \mathbf{c}_1 \mathbf{s}_1^T) [\mathbf{d}_3 + (\mathbf{S}_0^T \mathbf{d}_3) \mathbf{c}_0] \\
 &= (\mathbf{I} + \mathbf{c}_1 \mathbf{s}_1^T) (\mathbf{d}_3 + \alpha \mathbf{c}_0)
 \end{aligned}$$

Each level of update requires one vector dot-product to obtain the coefficient α followed by a vector add. The sequences \mathbf{s}_k and \mathbf{c}_k are stored to apply the updates.

4.4.3. Restarting Updates

Usually one will not continue with the initial approximation \mathbf{A}_0 until convergence. To save on update vector storage space, a number of iterations are done until space runs out, then the method is restarted. A restart consists of forming the Jacobian and factoring it and setting it to be the new \mathbf{A}_0 , at the current \mathbf{x} location.

In certain cases we do not wait for the update vector space to be filled but do an immediate restart. One such case is when phase changes occur, this is because the update vectors then relate to a different set of variables, i.e., the third variable might switch from \mathbf{S}_g to \mathbf{R}_s or vice-versa.

Restart may also be needed in the following situation. Dennis and Schnabel (1984) note that the Quasi-Newton direction, $s_k = -A_k^{-1}f(x_k)$, is not guaranteed to be a descent direction for the one-dimensional function $\frac{1}{2} \|f(x)\|_2^2$.

They mention that uphill directions do not appear often in practice and point out that an easy fix is to restart if any uphill direction occurs.

In tests on simulation problems it appears that uphill directions only appear if the Jacobian approximation A_0 is not adequate to the task, i.e. using too simple an approximation for difficult problems. The tests also showed that one may choose to ignore the occasional small uphill step without any detrimental effects. In practice it seems best to choose a good enough approximation so that the problem does not occur.

Another case where a restart may be necessary occurs when the denominator of the second term in Eq. 4.15 is very small. In such a situation one may decide not to perform the update and instead restart. This occurs very rarely.

Experiments showed that restarts occur mainly because of phase changes and running out of update storage space. These periodic restarts are beneficial in one way because the Jacobian is made up-to-date. Moreover the Quasi-Newton step after a restart is the same as the Newton step at that point so convergence is improved there. On the other hand the periodic restarts increase the number of times that the Jacobian has to be formed and factored.

4.4.4. Summary of the Method

The Broyden updates can be applied with relatively little cost. The principal penalty in using a Quasi-Newton method is the reduced convergence rate. At the k th iteration, we must obtain

$$d_k = A_0^{-1} f_k \quad (4.15)$$

and use the update sequence to get the final step. Potential savings arise from the fact that the same matrix or inverse is Used in Eq. 4.15 for several iterations during which the Jacobian also need not be formed. Also the method is accumulating secant information as the iterations proceed so there exists the possibility of starting out with a simple Jacobian approximation and building it up as we go. This is the approach used in optimization where we start with I .

4.5. TWO-PHASE EXPERIMENTS WITH THE QN METHOD

Early experiments were run to see if the QN method could take the approximation $A_0 = \text{diagonal } (J_0)$ and collect enough secant information to solve the nonlinear system of equations $f(x) = 0$. This was found to work for simple nonlinear equations and single-phase flow even in cases where Newton's method fails with the same approximation.

For two-phase problems it turns out that this approximation will not work and the method needs at least $A_0 = \text{block-diagonal } (J_0)$ each block being the 2×2 matrix of the oil and water equations at each grid point. This is probably because of a move from diagonal dominance to block-diagonal dominance.

The inverse of the block-diagonal matrix is also block-diagonal with each block inverted. For the 2×2 sub-matrices the elements of the inverse can be explicitly written out as algebraic expressions so the solution cost of Eq. 4.15 remains very low.

The change to a block-diagonal approximation proves adequate to simulate simple waterfloods. A quarter five-spot water flood simulation that had earlier been solved using Newton's method was run using this approximate Jacobian. While Newton's method averaged 3 iterations/time step (using a direct solver), Quasi-Newton took an average of 24 iterations/time step. However the cost of each QN iteration was so low that total time was still acceptable.

Figure 4.2 shows the total time taken as a function of number of grid blocks. There are two obvious ways to increase the number of grid blocks. One can keep the size of grid blocks constant and increase their number. This makes the problem easier to solve since the wells move further apart. The corresponding curve is labeled "Easy" in this figure.

Alternatively one can keep the overall size of the model constant and put more blocks into the same area. The size of blocks now decreases. This makes the problem harder to solve since transients are more severe in smaller blocks. This curve is labeled "Hard."

Figure 4.2 shows the expected difference in total times. It also shows how the Quasi-Newton method behaves with increasing problem size. One might wonder that since we are building up secants, do the number of secants needed increase as the dimensions of the problem increase? From Fig. 4.2 it is apparent they do not.

This is an important factor when using Quasi-Newton methods, for there is no guarantee that the search directions are orthogonal. If the number of search directions increase with problem size then the method would clearly be unacceptable.

Also shown is the linear increase line, with constant degree of difficulty, the simulation times may be expected to follow this. The kN^2 curve shows how an algorithm scaling with the square of the number of grid blocks would fare, the D4 algorithm, for example, scales approximately as N^2 (Aziz and Settari, 1979). The good performance of QN in comparison is due to the diagonal Jacobian being used and to the method's insensitivity to problem size. For the latter reason we expect that with a better approximate Jacobian, as long as the approximation involves $O(N)$ work, the overall algorithm will also scale linearly with constant degree of difficulty.

This property (if it could be guaranteed) appears very attractive since a algorithm that scales with N will eventually out-perform any algorithm that scales with a higher power of N no matter how efficient it may be. Unfortunately there is no guarantee and more powerful algorithms may also scale with N .

Figure 4.3 shows that a move from 2D to 3D also makes the problem more difficult, but the 3D curve is near-linear again.

Early parallel experiments were run on this waterflood model. Using a block-diagonal approximation for A_0 meant that the matrix solution could be trivially run in parallel. In addition the rest of the simulator was built to run in parallel. The details of the parallel simulator as well as the final comparison of the QN method are described in the next chapter.

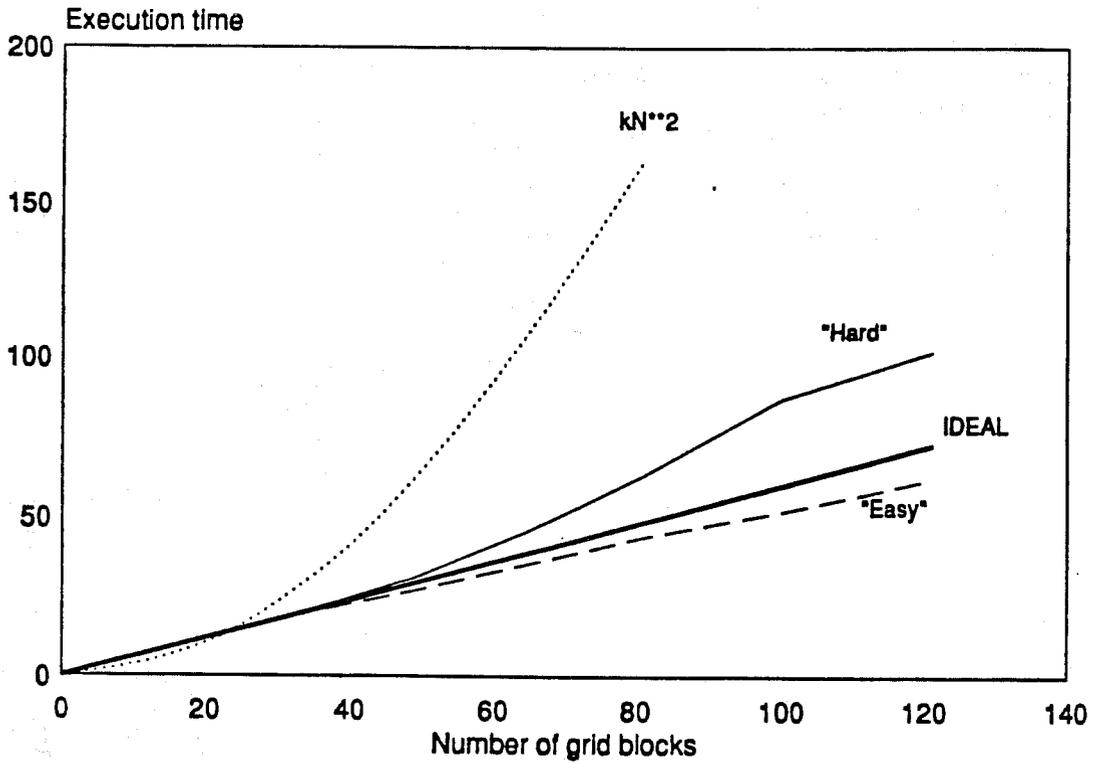


Figure 4.2. Total time on 2-D oil-water system.

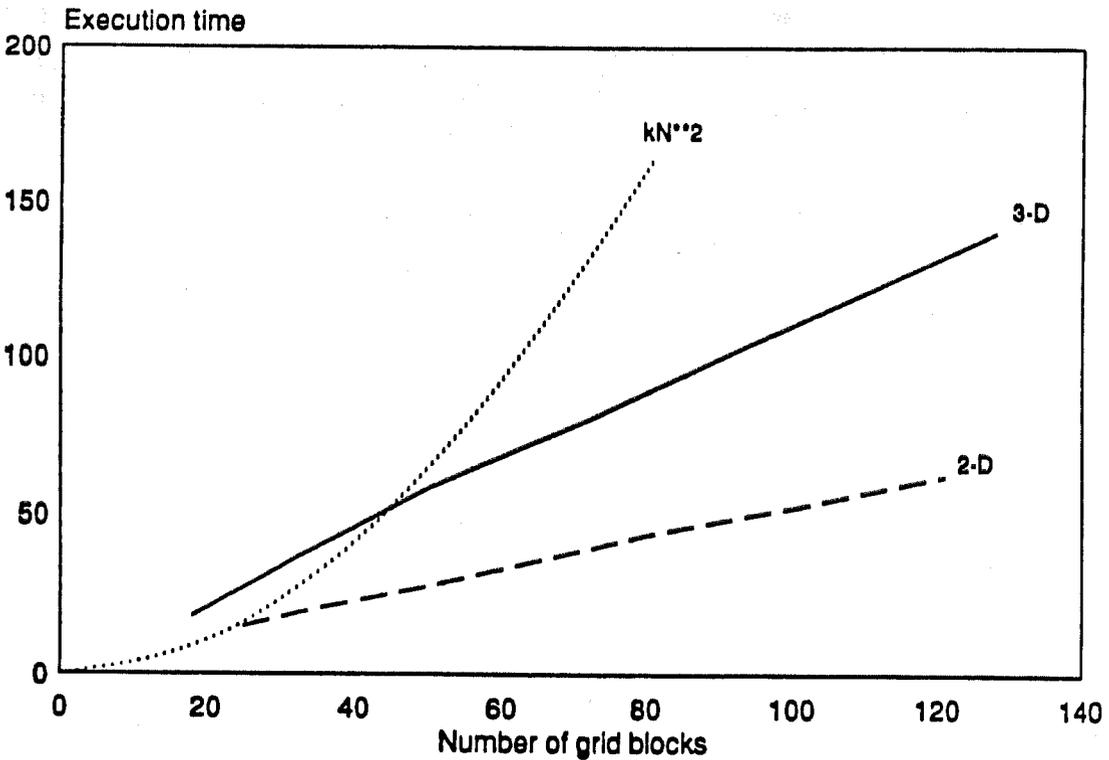


Figure 4.3. Time on 2-D and 3-D Oil-Water systems

5. PARALLEL RESERVOIR SIMULATION

Parallel computers provide a way to greatly increase the speed of reservoir simulation. To do optimization using numerical simulators the simplest way is to run several simulation jobs in parallel. However a more interesting problem is running one simulation in parallel on several processors, this is also useful for general purpose simulation tasks. A study on parallel reservoir simulation is detailed in this chapter as are final tests and evaluation of the Quasi-Newton method.

5.1. PARALLEL COMPUTING

The fastest supercomputers today use cycle times of a few nanoseconds. Since light itself travels only 11.78 inches in one nanosecond (electricity travels even less) such machines seem to be approaching physical limits. At such speeds physical distance to memory, and within a memory card, becomes a problem. Cost of components that can work at such speeds, especially memory, is also very high. Because of these factors further speed increments are more difficult and expensive.

Parallel computers capable of similar absolute performance can be made today at a much lower total cost. In addition it is possible, in principle, to enhance performance by simply adding more processors. It is likely that parallel computers will eventually greatly exceed the performance of conventional supercomputers, so they represent an important platform for reservoir simulation.

The form of parallel computers has not been standardized. Different approaches to solve the hardware issues have resulted in different types of parallel computers, e.g. message-passing, shared-memory, data-flow, systolic arrays and connectionist machines. Of these the first two types are the most promising for general purpose use.

Much effort is being spent on algorithms for parallel computation. It is relatively simple to make good use of a few processors working in parallel. In some cases this can be done automatically on existing serial code by the compiler and/or the hardware (Padua and Wolfe, 1986). However it is difficult to make effective use of a large number of processors working in parallel. It is this latter type of machine that holds the most promise and also presents the most difficulty in devising suitable parallel algorithms.

Usually one is forced to use an algorithm that is less than optimal for a serial computer. One cannot rely on the compiler to automatically detect and take advantage of concurrency in existing serial code; rather the code has to be explicitly written to run in parallel.

5.2. AMDAHL'S LAW

Even with parallel algorithms the achieved speedup is often not as good as expected principally because of Amdahl's law (Amdahl, 1967). Amdahl's law states that the time for a given computation T , is the sum of two components: a serial component S that must be performed in serial, and a parallel component P that can be performed in parallel. Therefore, on a single-processor machine the total time is:

$$T_1 = S + P$$

Now suppose that no time is lost on synchronization/communication etc. when running in parallel. Then the time for the same process on p processors will be:

$$T_p = S + \frac{P}{p}$$

and speedup is then:

$$\frac{T_1}{T_p} = \frac{S + P}{S + \frac{P}{p}} = \frac{1}{S + \frac{P}{p}}$$

which, in the limit of an infinite number of processors, approaches $1/S$. Even a small serial component can seriously affect performance, for example with a serial component of 5% the maximum possible speedup is 20. Actual speedup will be even less because of time lost in synchronization or communication between processors.

Because of Amdahl's law it has been widely believed that there was an upper limit of about 200 on the speedup possible in useful scientific computation.

Recently, Gustafson et al. (1988) broke through this limit by showing speedups of 1000 on 1024 processors on three different scientific problems. This speedup is based on a modified form of Amdahl's law. They also report the speedup by Amdahl's law as shown here to be around 500. This means that only 0.1% of the time was spent on the serial component and in communication, a remarkable feat since it means that a 16 hour computation on a uniprocessor machine would need only 1 minute of purely serial time.

5.3. PARALLEL MATRIX SOLUTION

The matrix solution step in simulation is the most difficult to parallelize. Conventional algorithms for matrix solution have high serial components and are unusable unless modified. Consider a triangular matrix, trivially solvable on a serial computer, it is a major challenge for a parallel computer. Since one element of the solution can only be obtained after the previous element has been obtained, there is an unavoidable serial component in solving this matrix. Some degree of parallelism is possible by operating on columns in parallel but gains are limited on the sparse matrices that arise in simulation.

Instead the matrix must be solved in a manner such that no processor has to wait for the result of another. Gustafson et al. (1988) solved their matrix problem using the point-Jacobi preconditioned conjugate gradient method (see Concus et al., 1976). This method is ideally suited for parallel computation, since the only serial component arises in the dot-products (i.e., vector inner products).

In addition by using point-Jacobi they were able to handle much larger problems than otherwise possible by storing only the main diagonal of the matrix. Since no matrix is stored they obtain the matrix-vector product that occurs in each PCG iteration by an additional function evaluation.

Their parallel speedup has been so spectacular that all other results (including those presented here) pale in comparison and this record will probably stand for a long time. However it remains to be seen whether such speedup will be possible with a more powerful matrix solution scheme.

Parallel matrix solvers have also appeared in reservoir simulation applications. Scott et al. (1987) tried various parallel SOR-type methods. They concluded that the *submatrix* method showed the best potential speedup. In this method each processor performs independent SOR iterations on part of the matrix. So various parts of the matrix are at different iteration levels. They showed a speedup of about 6.7 on 10 processors with this scheme.

Killough and Wheeler (1987) tried domain decomposition ordering as a preconditioner for an Orthomin (Vinsome, 1976) accelerated iterative method. The final form of their algorithm is a block Gauss-Seidel preconditioner combined with Watts (1971) line corrections. The grid is ordered such that the blocks corresponding to the domains are solved first, followed by blocks corresponding to the edges of the domains, finally the vertices between the edges are solved. This method was shown to compare favorably with current serial methods on ill-conditioned matrices but was slower by a factor of 3.2 when applied to a production reservoir simulator. They explained this by noting that the method seems to be linearly convergent after the first iteration, regardless of the condition number of the matrix. On ill-conditioned matrices current serial methods have difficulty converging but because of this linear decrease the method is competitive. In the reservoir simulator the matrices were not ill-conditioned so current serial methods, RS/ILU(0) (incomplete LU factorization with zero fill-in after reducing the matrix to half the normal size by red-black elimination - see, for example, Behie and Forsyth (1984) in particular, converged rapidly. They noted that each domain decomposition iteration is three times as expensive as a RS/ILU(0) iteration and this resulted in the poor performance. By finally limiting the solution to a single domain decomposition iteration they were able to get within 10% of the serial time.

Efrat and Tismenetsky (1986) used a biconjugate gradient algorithm, however, this method requires more work than a conjugate gradient type algorithm and has not since been used by others in the petroleum field.

5.4. OBJECTIVES OF CURRENT STUDY

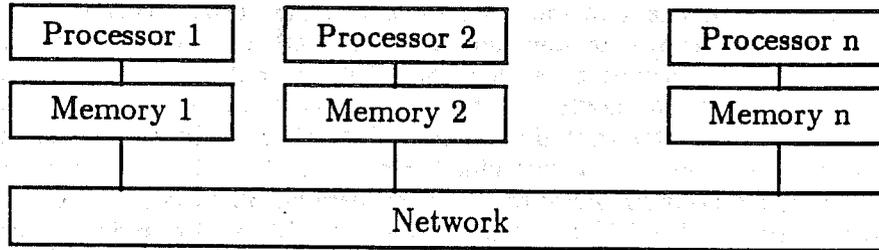
In addition to parallel matrix solution the parallel formation of the matrix equations also needs to be considered. The objective of this study was to construct a fully scalable (i.e., one capable, at least in theory, of achieving linear increase in speedup over any number of processors) parallel black-oil simulator and see what the issues involved are. In addition we also explored the use of Quasi-Newton methods for solving the nonlinear system of equations in parallel (and serial). In principle any parallel matrix solution scheme can also be used with the Quasi-Newton method and on the face of it the method appears attractive for parallel processing. We now briefly describe some of the issues involved in parallelizing a reservoir simulator, followed by a comparison of Newton and Quasi-Newton methods and conclude with tests of the parallel simulator.

5.5. SHARED-MEMORY AND MESSAGE-PASSING ARCHITECTURES

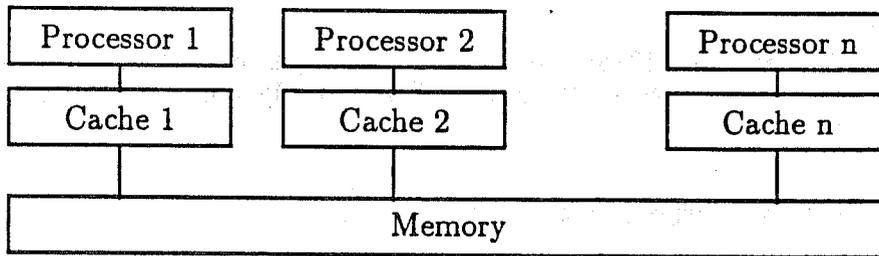
These are emerging as the most successful general purpose parallel architectures and are shown schematically in Fig. 5.1. Both usually operate as MIMD (Multiple Instruction Multiple Data) machines, i.e., each processor runs its own code on its own data.

On message-passing machines each processor has its own memory and processors communicate by passing messages over a network that ties together all the processors. The hypercube topology is usually used for the network because it keeps the interprocessor distance small (at most $\log_2(p)$ on p processors) while also keeping the communication traffic through a

PARALLEL MIMD ARCHITECTURES



Message-passing architecture



Shared-memory architecture

Figure 5.1. Shared-memory and message-passing architectures.

node small. Such machines can be built with a very large number of processors using relatively low cost memory and components.

Programs for these machines are quite a bit different from ordinary serial code, since messages must be passed to access global data. Messages move relatively slowly so one also has to spend effort on minimizing communication. In addition programs for such machines usually have a *host* process to schedule and control the parallel *node* processes, a concept never considered on serial code.

Shared-memory machines have a globally shared memory that can be accessed by all processors. Each processor usually operates out of a cache, but here the caches have to be more sophisticated than is usual in uniprocessor machines.

If a change is made to global memory by one processor, all caches must sense this, determine if they hold the memory location in cache and either invalidate the data or read it in. These machines suffer from the high bus traffic that results from multiple processors working with one shared memory. This architecture is therefore not suitable for a very large number of processors. On the software side a shared-memory machine is easier to program and communication between processors is very fast.

5.6. PROGRAMMING A SHARED-MEMORY MACHINE

In this study, runs were made on a 16 processor shared-memory MIMD machine, the Encore Multimax. The machine uses National Semiconductor NS32032 32 bit CPUs and NS32081 floating point co-processors. The operating system is similar to Unix. Physically each processor can access any of the shared memory, but from the logical point of view each processor has its own local memory and access to a global shared memory,

The shared-memory paradigm makes it relatively simple to write a simulator for this machine, the resulting code can run almost unchanged on a conventional serial machine.

The environment available for this study was most conducive to the C language so the simulator was written in this language. There are certain disadvantages in using C. Since the bulk of numerical code is in FORTRAN, little use can be made of existing code. C also does not handle multidimensional arrays as well as FORTRAN, specifically a subroutine/function cannot accept variable size arrays (unless they are one-dimensional). In FORTRAN, the declaration $A(N, N)$ allows any size two dimensional array to be passed. This is not possible in C so this makes everything a little bit harder to do since one-dimensional arrays have to be used throughout.

On the positive side C is very close to the machine, offers constructs like *do-while*, *switch*, user-defined types and records, preprocessing to allow selective compilation and other features typical of more modern languages.

Argonne National Laboratory's parallel processing package of process control, synchronization and communication primitives (described in Boyle, J., et al., 1987) provides the necessary parallel extensions to what is otherwise standard C language. The important primitives for shared-memory machines are named CREATE, LOCK and BARRIER and are briefly described below.

5.6.1. The CREATE Primitive

This primitive copies the entire program and data to a specified number of processors, starting them up at a specified subroutine. Each processor then runs its own copy of the program, using local memory for all variables except those explicitly allocated in global memory.

The simulator allocates space for all reservoir data and vectors and matrices in global memory. This is mainly for programming convenience, the code is designed so that each processor deals with a subset of the variables. Most variables like tables, loop counters and temporaries are kept local to each processor. A few (convergence flags, dot product results, global sums etc.) are explicit global variables and meant to be looked at or modified by any or all of the processors.

5.6.2. The LOCK Primitive

This primitive is used to control access to global variables. To see why this is needed, consider two processors incrementing a global variable. Suppose that the global variable is set at 21. Now if processors A and B each do an increment the variable should become 23. But suppose processor A reads in 21 and before it can write it out, processor B also reads in 21. Then A will write out 22 and so will B, giving an incorrect answer. To ensure correctness, A must be able to exclude B until it is done.

LOCK lets A make sure that B cannot access the variable until it has done its job and performed an UNLOCK. If two processors simultaneously try to get the LOCK only one will succeed (it shouldn't matter which one it is) and the other must wait. A typical application for these primitives is self-scheduling, where a processor determines what it should work on dynamically e.g.

```
LOCK();                /* obtain the lock */
my_index = global_index; /* get index to work on */
global_index++;        /* increment index */
UNLOCK();              /* let the lock go */
```

This sequence can be used in a simulator when doing table lookups for example. Each processor gets a grid block to work on and returns for another until all are done. Once a processor has the lock, all others trying to get the lock must wait so LOCKs reduce the amount of useful computation done.

In simulation the problem can be easily partitioned so this primitive is not used in the code. Instead we pre-schedule by assigning different parts of the reservoir (and corresponding parts of arrays and vectors) to each processor. Some global variables are used, but due to the nature of the preceding operations it is possible to avoid the use of LOCKs, either by allowing only a particular processor to modify the variable or by allowing all processors to modify the variable.

5.6.3. The BARRIER Primitive

This primitive is used to synchronize the processors. Upon coming up against a BARRIER statement a processor temporarily halts execution of the program. When all have halted at the BARRIER, it *opens* and all resume work.

The BARRIER primitive is useful in ensuring the consistency of data across all the processors. For example, suppose processors A and B are assigned grid blocks 1-22 and 23-44 respectively of a 1-D reservoir. Let's assume that the processors first go through their assigned blocks performing table look-ups and store the results. On a second pass they use these stored values and calculate the flow equation at each block.

If speed on the table lookups differ it is possible that one processor, say B, finishes before the other. The flow equation for block 23 will now be incorrect since it includes flow in from block 22 on which A has not yet done the lookup.

The solution, as shown in Fig. 5.2, is to put a BARRIER after the table lookups to ensure that all are done before work begins on the flow equations. It is also possible for each processor to maintain copies of the grid blocks surrounding its region and thus make sure of its own data by itself. This leads to redundant calculations and introduces some complexity in the code. So the simulator employs BARRIERS to make sure things are consistent.

Due to different times for table look-up, different outcomes of *if-then* statements, and presence or absence of reservoir edges and wells, each processor will be doing varying amounts of work even if all have the same number of grid blocks assigned to work on. This means that the load will never be perfectly balanced. Some processor will always take longer than the others to complete a given stage of the computation. So there is a slowdown at BARRIERS since the processors must wait there until the slowest one catches up. Some time is also needed for the system to actually execute the BARRIER statement itself. For these reasons BARRIERS tend to slow down the code.

USE OF BARRIERS

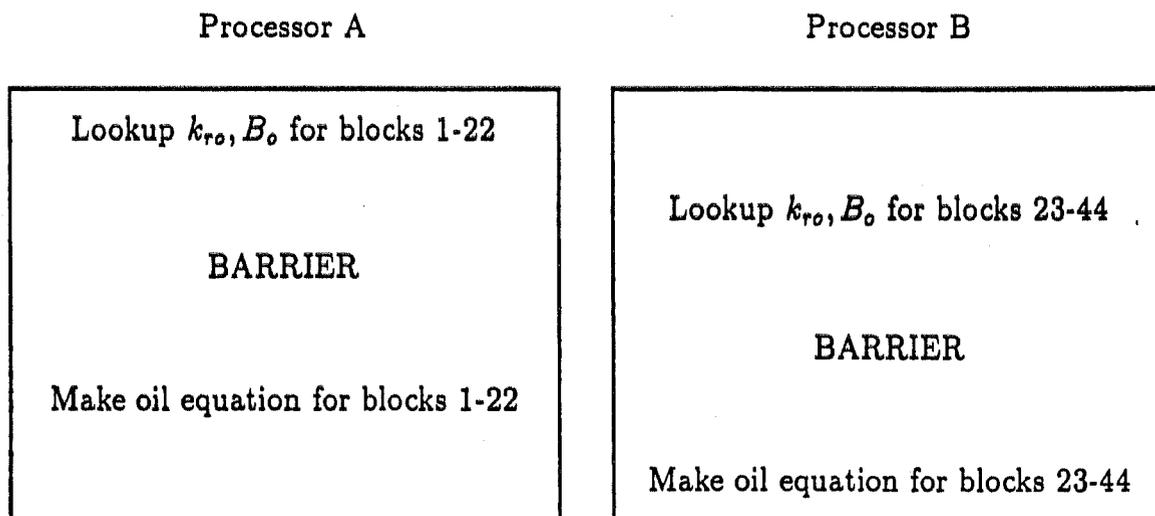


Figure 5.2. Use of BARRIERS on 2 processors.

In writing a parallel program one therefore aims for the coarsest granularity possible (i.e., long stretches of independent work given to the processors) and good load balance (i.e., equal amounts of work assigned to each processor) to reduce the time wasted at BARRIERS.

5.7. BUILDING A PARALLEL RESERVOIR SIMULATOR

Recall the fully implicit formulation of the previous chapter:

$$T_{i+\frac{1}{2}}[p_{i+1} - p_i - \gamma_{i+\frac{1}{2}}(z_{i+1} - z_i)] - T_{i-\frac{1}{2}} [p_i - p_{i-1} - \gamma_{i-\frac{1}{2}}(z_i - z_{i-1})] - \frac{V}{\Delta t} \left[\left[\frac{\phi S_1}{B_1} \right] - \left[\frac{\phi S_1}{B_1} \right]^n \right] - Q_1 = 0 \quad (5.1)$$

This can be written as $f(x) = 0$. So the major tasks in parallel simulation are to form $f(x)$, and its Jacobian $J(x)$ and then solve the associated matrix problem, all in parallel.

Scott et al. (1987) talk about pipelining these separate tasks so that one processor could, for example, begin work on making f before the matrix solution is complete. However this may not be practical because in order to obtain the maximum throughput we want to always use perfectly balanced *parallel* techniques everywhere. So, in principle, no processor will finish on a task before the others.

The approach used in this study is to complete one task in parallel on all processors before going on to the next task. For good load balance each processor is assigned an equal number of grid blocks and equations to work on. When the number of grid blocks cannot be equally divided the remainder is distributed one by one among the first few processors (e.g., for 14 blocks on 4 processors the assignments are 4 4 3 and 3 grid blocks each).

Figure 5.3 shows the program flow chart of the parallel simulator. The arrows represent major BARRIERS. These ensure that each processor will be working on identical code sections at the same time, each on its own set of grid blocks or equations.

The only exception to this rule is processor 0, which is assigned the task of printing out results (parallel I/O is not possible on this machine), and also the task of updating global variables like dot-product results, convergence etc. All instances of the program check to see if they are on processor 0 and the one that passes proceeds to perform the task while the others either wait for the result or continue work until the next BARRIER.

5.7.1. Parallel Construction of f and J

Most of steps required to construct f and J can be run independently in parallel. The tables are replicated on each processor so table lookups go through without any interference between processors. They are followed by a BARRIER to counter speed mismatches.

For maximum performance in the construction of f and J , special treatment is needed when running in parallel. If used as written, the finite difference equation (Eq. 5.1) contains redundant calculation, for example at block 2 one of the flow terms is $T[p_2 - p_1 - \gamma(z_2 - z_1)]$. However the identical term has already been evaluated when

PROGRAM FLOW CHART

(Arrows correspond to major synchronization points)

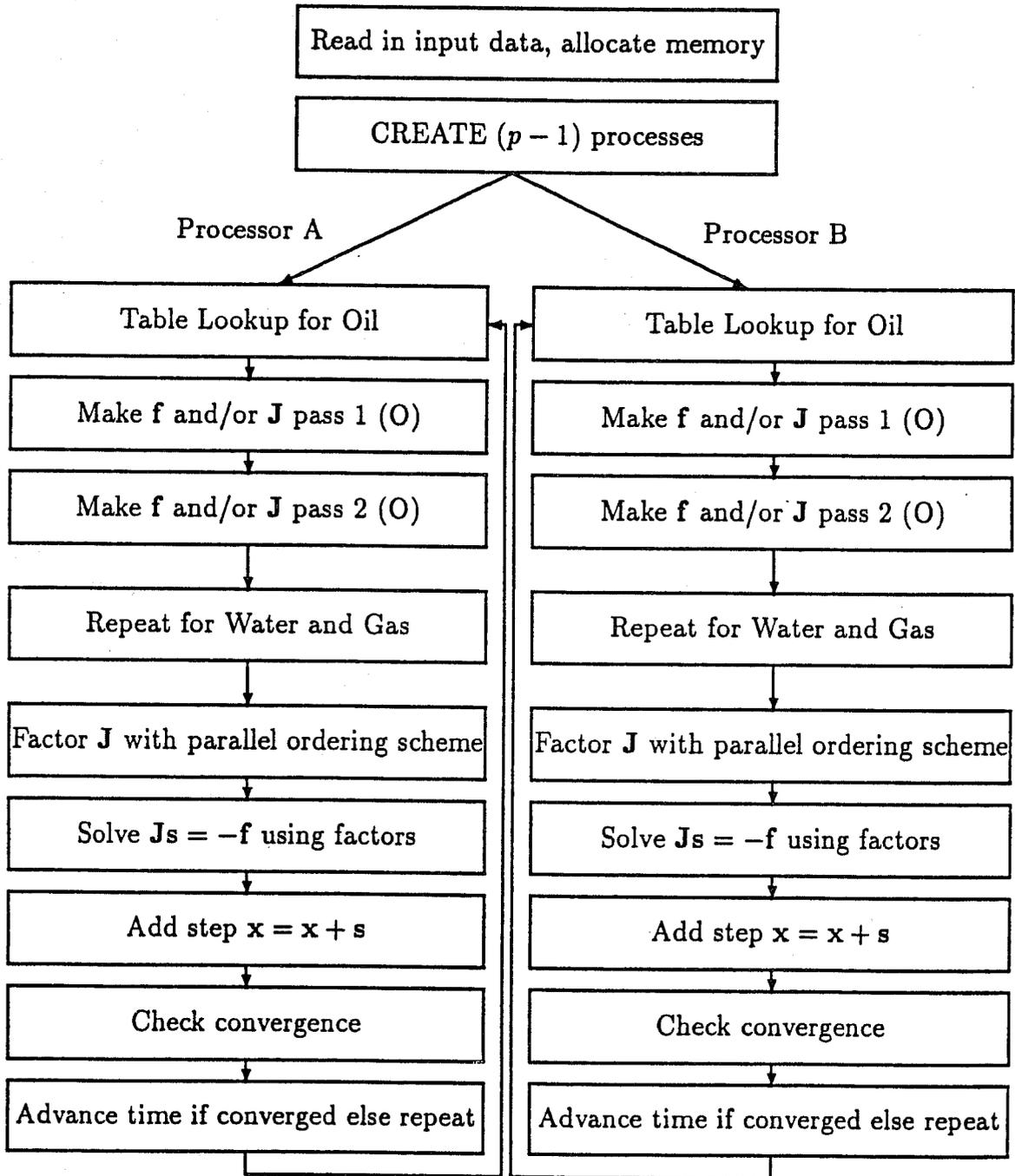


Figure 5.3. Program flow chart on 2 processors.

processing Block 1. The transmissibility at the interface will be identical for both blocks regardless of which is upstream. So this term and its derivative needs to be evaluated only once.

In two dimensions we need only calculate the flow rate and its derivative for a block and its two downstream neighbors, $(i + 1)$ and $(j + 1)$, and add these to the neighbors as flows from $(i - 1)$ and $(j - 1)$. So by the time a block's turn comes up half the work has been done. This is easy to implement in a serial computer but some care is needed on a parallel computer. Because the flow rates and derivatives are *summed* on the main diagonal of the Jacobian and in f there is a chance for error during the summation if different processors try to add in flow contributions to the same grid block at the same time.

These contributions occur too frequently to bracket each one with a LOCK and UNLOCK sequence. Extra blocks at edges will solve the problem but this introduces redundant calculations. The chance of error is avoided if the processors store (not sum) these contributions. Storage space is available in the vectors used by the Orthomin acceleration scheme.

Constructing the Jacobian and f is then a two pass process as shown in Fig. 5.4. In the first pass the processors store flow terms and derivatives at each block and at its neighbors. A BARRIER at the end makes sure that all are done. Finally, in a second pass the processors run down each block, adding on the previously stored entries (with the appropriate change in sign), add accumulation terms and wells to complete the Jacobian and f .

CONSTRUCTING f AND J

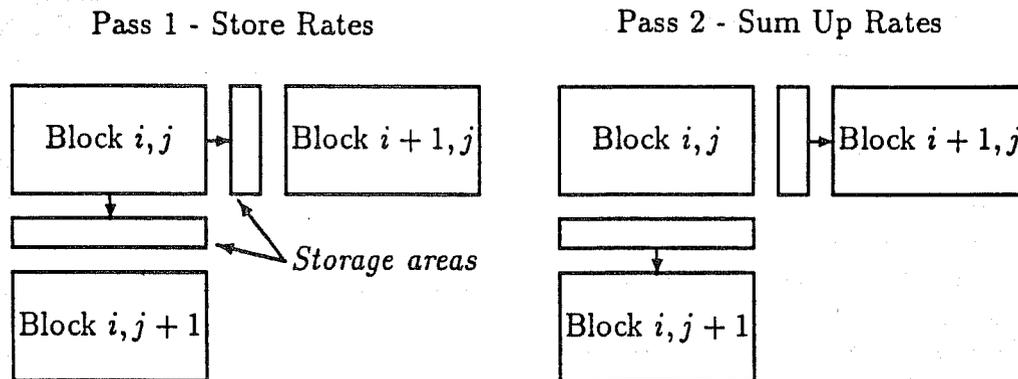


Figure 5.4. Parallel construction of f and J .

5.7.2. Parallelizing Vector Dot Products

Vector dot-products have significant serial component. Each processor can, in parallel, perform dot-products on part of the vectors, but the partial results must then be summed in serial to obtain the final dot-product. A code example is:

```

BARRIER(); /* wait for consistent arguments */
MakePartialDotProducts();
BARRIER(); /* wait for all to finish */
if (ProcessorID == 0)
    Global->DotProduct = SumUpResults();
/* sum up partials in serial on processor 0 */
/* and store the result in a Global variable */
BARRIER(); /* wait for serial sum to finish */
    
```

This code requires 3 BARRIERS and the serial sum. Some time can be saved by careful coding. For example, by checking the places the routine is *called* from we can remove the first BARRIER from this routine and place it before the *call* only where needed.

It is also possible to save time on the serial step by using more than one processor. If the partial results are stored in a vector, one can recursively add odd elements to even elements in parallel, halve the vector length and repeat until this length is one.

With a large number of processors this saving is useful for it reduces the floating-point time to $\log_2(p)$ instead of p . This is important in the 1024 processor machine used by Gustafson et al. (1988) since $\log_2(1024) = 10$.

With 16 processors the serial sum requires 16 floating point add/accumulates. The faster method reduces this to 4 floating point add/accumulates but requires integer comparison, decision making and BARRIERS after each add/accumulate, so it is not attractive for this machine.

With the limited size of the problems that could be run on this machine, even this small number of serial instructions has quite an effect. For example, consider the dot product of two 800 element vectors. On 12 processors, 67 multiply/accumulate instructions are enough for an ideally parallel operation. Instead we need $67 + 12$ instructions and the 3 expensive BARRIERS - so speedup is much less than 12.

The dot-product problem can also be solved using the LOCK/UNLOCK primitives. When each processor is done with its partial result it can get the lock on a global sum variable, add on its result and release the lock. Speedup is not improved since 16 sums still have to be done, one after the other but this time with the added overhead of a LOCK/UNLOCK pair surrounding each sum. There is also the need to zero the sum variable initially and to wait for all processors to complete before passing the result back.

5.7.3. Miscellaneous Linear Algebra Routines

Common routines like vector adds, copying vectors etc. (the most heavily used is the addition of a scaled vector to another: $s = ax + y$) are handled by a small library written to do the operations in parallel. Each processor is pre-assigned a subset of the vector elements to look after so the routine simply executes the operation on the subset in its charge.

By calling on subroutines to do these basic tasks, the main program can be written as if it were for a serial machine.

5.7.4. Globally Shared Variables

Certain flags and sums require global variables. For example, to calculate oil in place each processor makes a partial sum and stores the answer in its slot of a global oil-in-place vector. When all the processors are done, Processor 0 sums up the partials as in the dot-product.

For automatic time step control each processor stores the maximum change in P and S_i on its grid blocks. Once again Processor 0 finally obtains the overall maximum changes.

For convergence tests, warnings about table look-ups being out of bounds and for well rates the processor involved directly changes the associated variables.

5.8. PARALLEL EXPERIMENTS WITH THE QUASI-NEWTON METHOD

Early parallel experiments were run on the waterflood model used as a test for the Quasi-Newton method. The block-diagonal approximation for A_0 allows the matrix solution to be trivially run in parallel. In addition the rest of the simulator was built to run in parallel as detailed above.

So without addressing the question of when the QN method is more efficient we have a fully parallel simulator to experiment with.

During these experiments reservoir block sizes were kept constant, while their numbers were changed to present different sizes of problems. Due to memory constraints on the machine the problem size was limited to 400 grid blocks.

Figure 5.5 shows clearly the benefit of moving from a serial computer to a parallel computer. Compared are the time on 1 processor with the time on 12 processors. Notice how the total time needed increases much more gradually with problem size on a parallel machine. The figure also shows the previously noted effect of Quasi-Newton methods, the serial work scales linearly with problem size with a diagonal Jacobian approximation.

Figure 5.6 compares the speedup at various problem sizes to the ideal speedup. Note that for the 5×5 problem there is a decrease in speedup as the number of processors increases. On such a small problem each processor is doing very little work before coming up against synchronization points at BARRIERS. This work shrinks as the number of processors increases until finally the time lost at BARRIERS overcomes the speedup obtained by running in parallel.

It is obvious that as the problem size increases, the better the speedup obtained, in fact this is one of the standard conclusions of researchers in parallel computing. Figure 5.7 shows the speedup on 4 processors with various problem sizes. From 100 grid blocks onwards speedup is close to ideal. The speedup curve still trends upwards at 400 grid blocks so it is likely that speedup can get even closer to ideal. This figure shows that on a small number of processors it will be relatively simple to get very close to ideal speedup.

Figure 5.8 shows the speedup on 12 processors with various problem sizes. At small sizes we see the detrimental effect of BARRIERS; observe the speedup of only 2.5 on the 5×5 problem. Speedup is disappointing even at 400 grid blocks. At this size each processor is still only dealing with 33 or 34 blocks. It appears that the overall problem size would have to be at least 2-3 thousand blocks for this problem before the times could begin to approach ideal speedup.

As the number of processors go up this threshold will also rise (the way it has done when moving from 4 to 12 processors). In general, problem sizes have to be very large in order to get good speedup, for instance Gustafson *et al.* (1988) got their best results with 2 million finite elements (2048 elements on each of 1024 processors), a problem that would have taken 20 years to solve on a single processor.

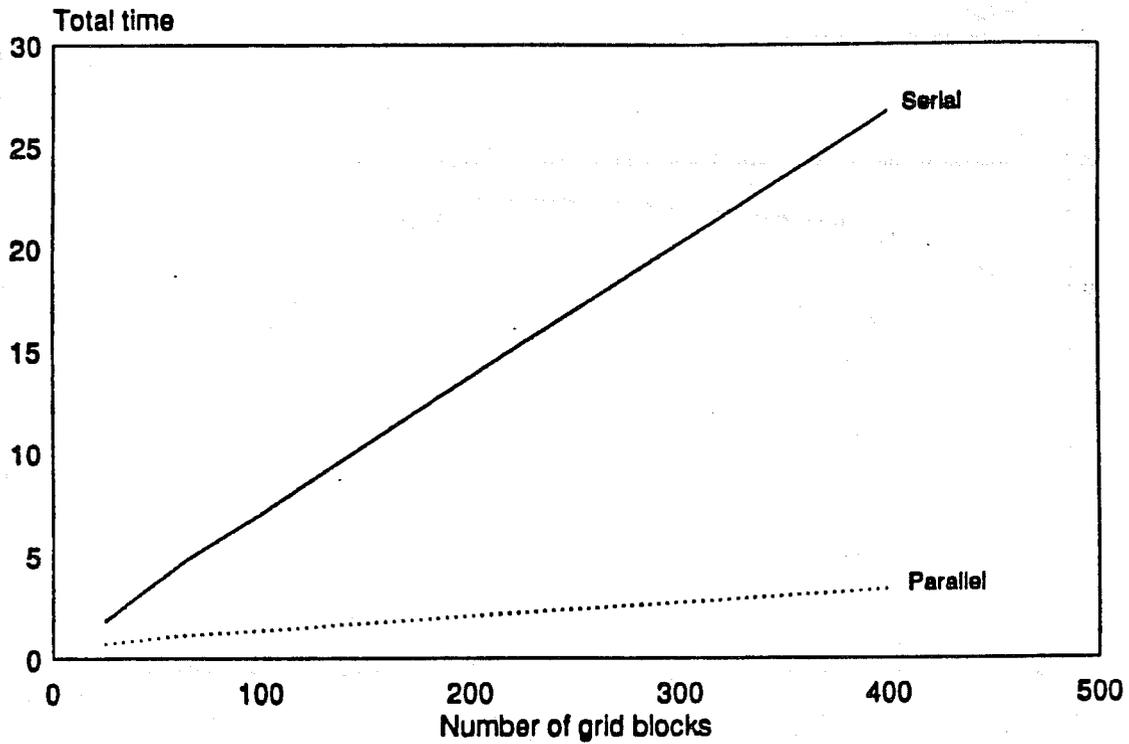


Figure 5.5. Time on serial vs time on parallel.

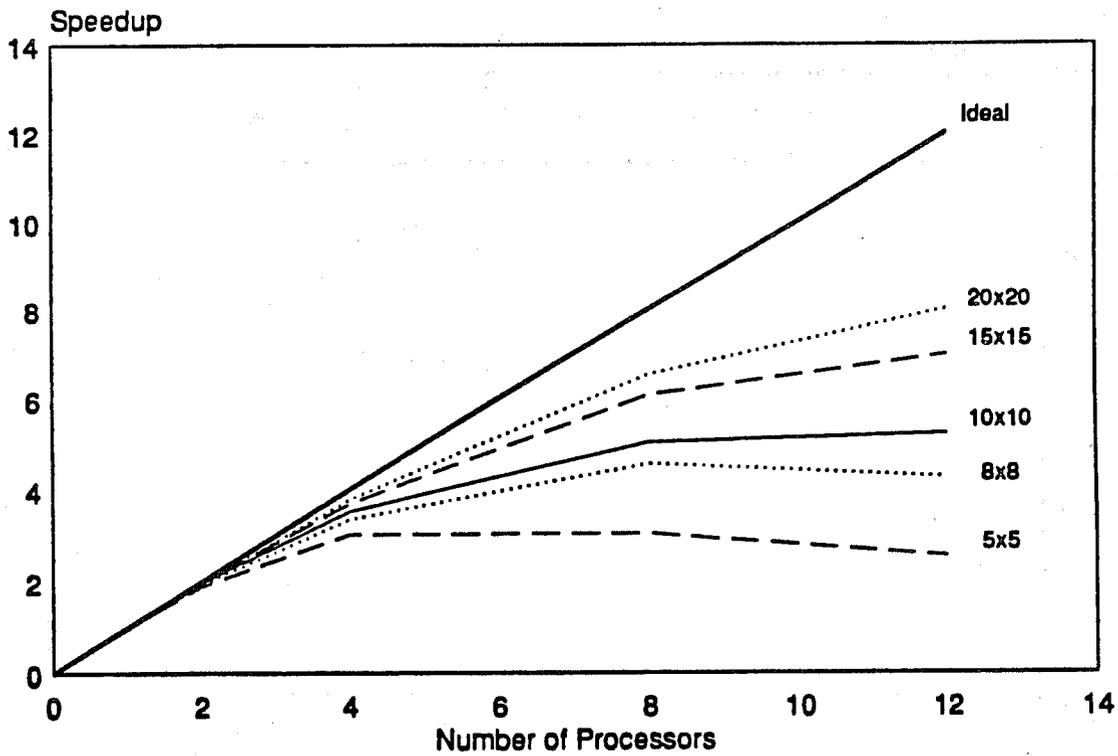


Figure 5.6. Speedup vs problem size.

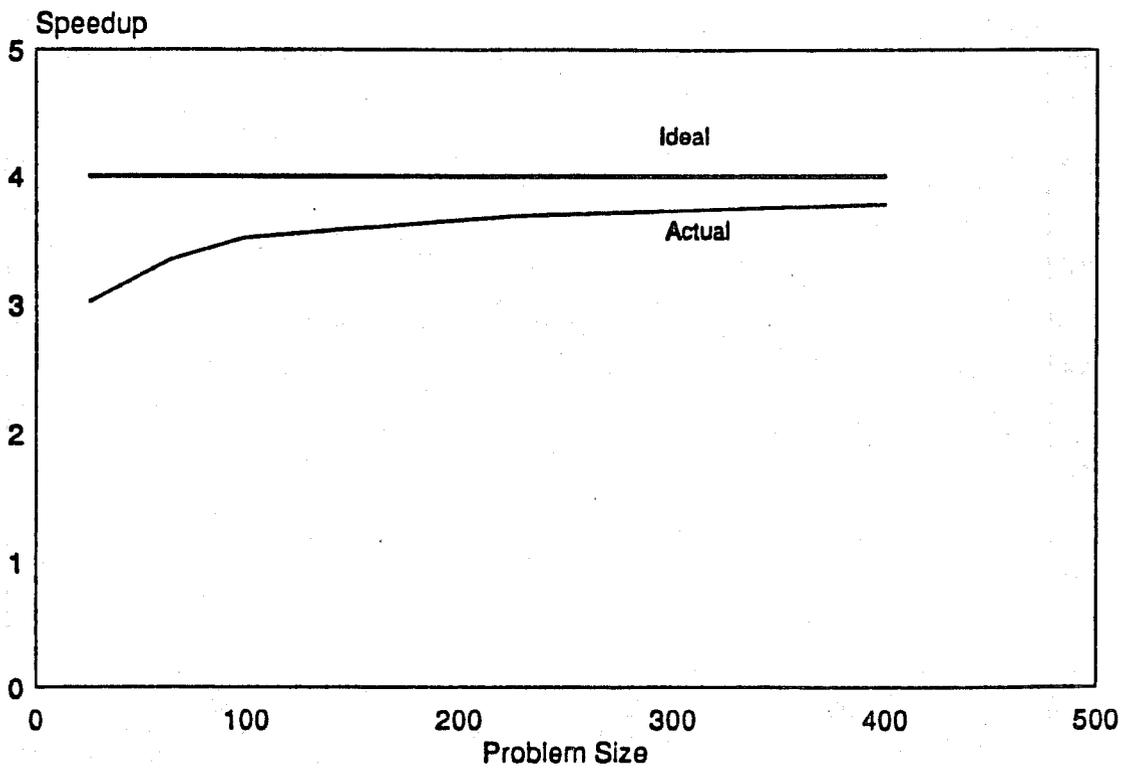


Figure 5.7. Speedup on 4 processors.

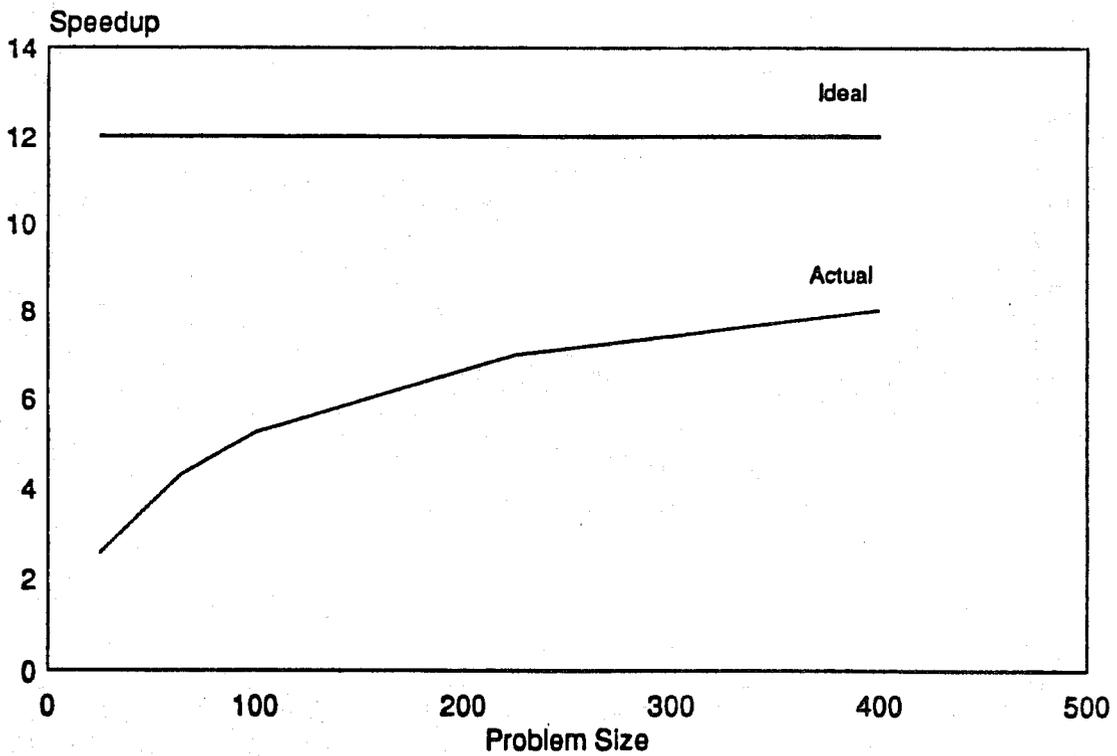


Figure 5.8. Speedup on 12 processors.

5.9. COMPARING NEWTON AND QUASI-NEWTON METHODS

We now compare Newton and QN methods on three phase simulation of a standard test problem on a serial machine.

5.9.1. Variable Substitution

Recall that the Quasi-Newton method holds the accumulated secant information in update vectors. This works well with the two phase oil-water system but three phase simulation requires the treatment of gas. The unknown variable that accounts for gas changes from S_g when free gas is present, to R_g when all free gas disappears and vice versa. The Quasi-Newton update vectors relate to the set of variables at the start of the time step. On variable substitution these vectors become outdated so the method must be restarted with the new set of variables. This can be costly if factorization costs are high so ideally one would like to avoid this forced restart, if possible.

The pseudogas concept, where a small amount of gas is always present, avoids variable substitution but, as shown by Forsyth and Sammon (1984), has a greater material balance error. To avoid variable substitution and material balance error one can use a single variable for the total gas in a grid block, i.e.:

$$\bar{S}_g = \frac{S_g}{B_g} + \frac{R_s S_o}{B_o} \quad (5.2)$$

Given \bar{S}_g it is relatively simple to find S_g or R_s . If $\bar{S}_g > R_s S_o / B_o$, where R_s and B_o are the values for a saturated oil at the block pressure, then free gas is present and S_g can be determined from Eq. 5.2. Otherwise $S_g = 0$, and R_s can be found using a 1-D Newton scheme and the B_o lookup table [the Newton scheme is needed because $B_o = B_o(R_s, P_o)$].

So this one variable can represent both R_s and S_g and restarts should no longer be necessary. However there are large changes in the Jacobian when gas evolution or solution takes place. In a Quasi-Newton method the Jacobian will have been constructed before the gas phase appeared or disappeared and the updates must change it to reflect the new status (similar to Mattheis and Strang's (1979) handling of a change from elastic to plastic in structural finite elements).

Tests show that the Quasi-Newton updates can work when gas comes out of solution but do not work when free gas goes into solution. The method seems unable to update a Jacobian constructed with free gas present, to account for disappearance of the gas. This is probably due to the big decrease in effective compressibility of the block on such a change.

5.9.2. Test Conditions

A standard test case, the first SPE comparative solution project (see Odeh, 1981) was the basis for test runs. This is a $10 \times 10 \times 3$ simulation model with fairly high permeability contrasts, high rate gas injection and high rate oil production. For benchmark purposes a ZXY ordered band solver was also tried on this problem. This ordering has a smaller bandwidth than standard ordering.

All cases used the same nonlinear convergence test as follows. Convergence was acceptable when either:

$$|x(i)| < 1.0$$

$$\& |s(i)| < 1 \times 10^{-5}, \forall i = 1, \dots, N$$

or

$$\left| \frac{s(i)}{x(i)} \right| < 2 \times 10^{-5}, \forall i = 1, \dots, N$$

where x is the vector of unknowns and s the step vector.

The first test appears lax but usually pressure converges slower than saturations (this is the basis for the COMBINATIVE method of Behie and Vinsome, 1982) so the second test will usually apply and prevent premature acceptance. The first test is used to avoid the occasional occurrence of a tiny gas saturation appearing in the denominator of the second test, magnifying an acceptable step, and not allowing the iterations to complete. The actual numbers are *ad hoc* but appear to be adequate, a later 10 year run on this problem with a linear solver tolerance of 0.01 gave zero material balance error for oil and 0.0005% error for gas (error being expressed relative to quantity produced).

All runs simulated the first 100 days of this test, unless otherwise specified. Time steps started at 1 day and increased by a factor of 1.5 thereafter. No time step cuts were necessary.

5.9.3. First Order Techniques

With gas in the problem the block-diagonal approximation for A_0 is no longer adequate and must be improved. Dubois et al. (1979) discuss a method for improving on inverses using a truncated Neumann series approximation for A^{-1} . To solve $Ax = b$ they define the splitting $A = M - N$. Then since $A = M(I - M^{-1}N)$, and using the Neumann series expansion for an inverse

$$A^{-1} = (I - M^{-1}N)^{-1}M^{-1}$$

$$= \left[\sum_{i=0}^{\infty} (M^{-1}N)^i \right] M^{-1}$$

$$= \left[\sum_{i=0}^{\infty} (I - M^{-1}A)^i \right] M^{-1}$$

The sum is truncated to obtain an approximate inverse - p terms give the approximation M_p^{-1} . They show that one iteration with M_p^{-1} is equivalent to p iterations of

$$x^{(k+1)} = x^{(k)} + M^{-1}[b - Ax^{(k)}]$$

When M is the diagonal of A this is the iteration scheme for the first order (i.e., without acceleration) Jacobi method.

So one can either improve A_0^{-1} for the QN method using a Neumann series or equivalently solve the equation $A_0 d_k = f_k$. At this point we face a fundamental problem with the Quasi-Newton method. The principal advantage of the method is that the same inverse can be used for several iterations. Ideally we would like use this improved inverse rather than solve the equivalent equation. However, the inverse will not be sparse so we cannot afford to store it, instead the equation must be solved at each iteration. This reduces the possible gains from the method if the matrix equations are solved iteratively.

It turns out that at least *z-line* Jacobi iterations are needed for this problem because of the high vertical transmissibility. To do this the grid blocks are numbered along vertical lines and M is the (block) tri-diagonal part of A .

Table 5.1 shows a comparison of Newton and Quasi-Newton schemes under the standard test conditions. Cases A through D show *z-line* Jacobi iterations done to various convergence tolerances. The iterations continue until either the l_2 norm of the residual ($f_k - A_0 d_k$) has dropped below a specified fraction of the l_2 norm of the initial residual (i.e., of f_k), or a maximum number of Jacobi iterations have been done. In Case E, the ZXY band solver is used. Time reported is the total time for the simulation.

Case E shows how the QN method can save with direct solvers, because in most iterations it can get away with just a forward-elimination followed by back-substitution on the LU factors of the Jacobian matrix, substantial savings in time are possible.

Case A shows the effect of solving iteratively to high accuracy. Here the linear convergence tolerance is tight, resulting a good solution - Newton's method takes the same number of iterations as with the direct solver. The Quasi-Newton method fares poorly in this case, taking more iterations than Newton's method. This is because the convergence of a Quasi-Newton method is at most superlinear while Newton's method converges quadratically.

Cases B through D shows the effect of relaxing the convergence tolerance. Total time drops because of the reduced time spent on the matrix solution. But notice also that as the approximation gets worse the QN method begins to perform better. The nonlinear iteration count for Newton's method begins to increase, indicating a drop in convergence rate from quadratic. The convergence rate of the QN method falls less dramatically, until in Case D it requires fewer nonlinear iterations than Newton's method. In this last case Newton's method takes 44% longer and also produces a less accurate answer; material balance is orders of magnitude worse than that obtained with the Quasi-Newton method.

From this table it is clear that the QN method can make use of the secant information that is accumulated. Of course, as Case A shows it cannot do better than superlinear and so it will not converge faster than Newton's method with an exact solution. However for the same reason that it does well with the approximate solutions shown in this table, we expect that it will do well when used with an approximate Jacobian and later experiments confirm this.

5.9.4. Accelerated Techniques

Orthomin (Vinsome, 1976) is the most popular acceleration technique in reservoir simulation and is the equivalent of the preconditioned conjugate gradient method for nonsymmetric matrices. It is based on generating a series of A-orthogonal search vectors and minimizing the residual in the direction of the search. At the k th iteration, if the search vector is Aq_k and the residual is r_k , then the residual is minimized in the search direction by choosing the step length ρ such that $\|r_k - \rho Aq_k\|_2$ is minimized.

Table 5.1. Comparison of direct and first order iterative methods

Case	Method	Residual Tol.	Maximum Jacobi Itns	Number of Nonlinear Itns	Number of Jacobi Itns	Time sec
A	Newton	0.001	100	43	2408	251.1
	QN			60	3249	331.8
B	Newton	0.01	50	51	1680	182.7
	QN			70	1623	174.9
C	Newton	0.1	20	78	1133	139.1
	QN			81	774	94.1
D	Newton	0.2	15	92	946	126.1
	QN			81	708	87.4
E	Newton	Band Solver		43		1225.5
	QN			58		545.6

Because the search vectors are orthogonal the method is guaranteed (except in pathological cases) to be able to span the vector space in its search for the solution. Furthermore by minimizing the residual the method, in effect, acts like a direct method at each orthogonal step. The combination of orthogonalization and minimization makes the method extremely powerful.

Since QN must also use the same acceleration method to match Newton's time, a QN method accumulates secant information in precisely the same search direction. So the only extra information picked up by Quasi-Newton is the difference between the linear model and the nonlinear model in *one* direction. On the other hand the effectiveness of the minimization is hampered by the QN scheme since the old Jacobian is used for A. So the superlinear versus quadratic convergence rate tradeoff again applies when comparing the two methods.

Table 5.2 shows the effectiveness of the Orthomin acceleration scheme and the performance of the QN method with various linear solution schemes. In this and all subsequent tests the linear tolerance used was 0.1 and the number of linear iterations limited to 15, unless noted otherwise.

The columns correspond to, in order, number of iterations (nIt), number of factors (nF), number of Jacobian evaluations (nJ), number of forward eliminations/back solves on the previously factored Jacobian (nS), number of Watts' line corrections (nW), time for 100 day simulation, material balance error for oil in barrels per million barrels produced, material balance error for gas in MCF per 100MMSCF produced.

The rows correspond to the linear schemes which, in order, are z-line Jacobi, Red-Black Gauss-Seidel (RB GS), RB GS with Orthomin acceleration and RB GS with Orthomin acceleration and Watts (1971) line correction. These are detailed in the next section.

The table shows that with the first-order iterative schemes the QN method can do better than Newton's method but with acceleration the QN method is slower. However this does not rule out the QN method, for it requires fewer factorizations (nF) and Jacobian evaluations (nJ) than Newton's method. More solve steps (nS) and function evaluations are required by QN. Tests show that the time to form the function is about half the time needed to form the Jacobian which itself is not very expensive in black oil simulation. So we omit costs of forming f in further discussions. The relative merits of the two methods will depend upon how expensive it is to form the Jacobian and do the factorization step, relative to the solve step.

The factorization step consists of converting the preconditioning matrix, $M \approx J$, into its factors, i.e. the upper triangular matrix, U, and the lower triangular matrix, L, such that

$$M = LU$$

The solve step consists of solving $Mx = b$, by forward elimination

$$Ly = b$$

followed by backward substitution

$$Ux = y$$

and acceleration in iterative techniques.

Table 5.2. Comparison of iterative methods

	nIt	nF	nJ	nS	nW	Time sec	Oil Error	Gas Error
Newton								
zLine Jacobi	99	99	99	1214	0	87.8	1408	582
Red-Black GS	75	75	75	696	0	64.9	1408	224
RB GS + ORTH	55	55	55	169	0	22.6	192	69
RB GS+ORWatt	52	52	52	62	242	22.7	0	0
Quasi-Newton								
zLine Jacobi	91	28	28	940	0	64.7	0	10
Red-Black GS	85	28	28	533	0	49.8	0	9
RB GS + ORTH	81	25	25	229	0	26.7	304	117
RB GS+ORWatt	71	21	21	80	299	24.5	16	0

The relative cost of these steps depends upon the linear techniques used:

- With a direct method there is a clear case for the QN method since $M = J$, and the factorization step will be much more expensive than the single solve step.
- With iterative schemes the performance depends on the form chosen for M , and the method used for the factorization. For example we use a tridiagonal M , so the factorization cost is very low. If a denser M were used, the cost of LU factorization will go up.
- Incomplete LU factorizations set $M = J$, but do not factorize exactly. Instead they save time by allowing only a certain level of infill terms to appear in the factors. The cost of factorization increases with the level of infill allowed.
- The solve step in iterative schemes requires one $\sqrt{2em}$ or more iterations of forward elimination, back substitution and acceleration. If the number of iterations is excessive then this cost will be much higher than the factorization cost so the QN method will not be competitive.
- The ideal situation for QN seems to be a preconditioner with a high setup and factorization cost, followed by one solve step per nonlinear iteration (this is similar to what Killough and Wheeler (1987) needed to do in order to get acceptable performance). Usually such preconditioners will require very few solve iterations anyway (in the limit, direct methods require only 1 iteration) so they will be more conducive for the QN method.

With the z-line Gauss-Seidel preconditioner used here the matrix is tridiagonal and this can be factorized very quickly. The solve step requires one or more iterations of forward-elimination, back-substitution and acceleration all of which put together are more expensive than the factorization, so little can be expected from a QN method with this preconditioner on this problem. For this reason the Watts aided results in Table 5.2 show that the Quasi-Newton method takes longer than Newton's method in spite of using less than half as many factorizations and Jacobians.

5.9.5. Effect of Number of Updates

All the previous runs have restarted the QN method after 7 updates or on a phase change. This number could be reduced to improve convergence. Table 5.3 shows the effect of varying the number of update vectors. During this simulation several update restarts occur due to variable substitution so each iteration has not been run to the number of updates allowed. Still some obvious trends can be observed. Nonlinear convergence can be improved by reducing the length of the update sequences but not to a very great degree; observe the small drop in number of iteration from 71 to 69 in moving from QN(8) to QN(5), the number in parenthesis being the maximum number of updates performed before restarting. The number of factorizations increases more rapidly, from 20 to 25 for the same case and total time decrease only slightly. Note that the performance for both methods is markedly better with Watts' line correction, particularly in material balance.

Table 5.3. Comparison of number of updates

	nIt	nF	nJ	nS	nW	Time sec	Oil Error	Gas Error
Red-black z-Line Gauss-Seidel Orthomin								
Newton	55	55	55	169	0	22.6	192	69
QN(8)	82	23	23	233	0	26.9	160	66
QN(7)	81	25	25	229	0	26.7	304	117
QN(6)	81	26	26	232	0	27.4	288	117
QN(5)	78	27	27	221	0	26.3	176	72
Red-black z-Line Gauss-Seidel Orthomin + Watts								
Newton	52	52	52	62	242	22.7	0	0
QN(8)	71	20	20	80	299	24.5	16	0
QN(7)	71	21	21	80	299	24.5	16	0
QN(6)	70	24	24	80	297	24.2	0	3
QN(5)	69	25	25	78	292	23.7	0	0

5.9.6. Effect of Simulation Parameters

The previous discussion has compared the two nonlinear methods on a standard test case and showed that with a tri-diagonal preconditioner the Quasi-Newton method cannot do better than Newton's method on the test case.

Some further tests were run to see under what conditions the QN method may be appropriate. The first test is one simulating depletion in a solution gas drive reservoir. Dimensions and physical properties are the same as before but this time the gas injector is nearly shut in (injecting at 1mmscfd instead of 100mmscfd). The reservoir is just above bubble point so evolution of gas is taking place in nearly every time step as the producer draws down the reservoir pressure. Table 5.4 shows the results on this test.

This problem is easier than the standard test case and the nonlinear iteration counts (nIt) clearly reflect it. Because the problem is less nonlinear the QN method requires only a few more iterations than Newton's method. Where previously the QN method needed 71 iterations versus 52 for Newton, now the numbers are 39 to 38. So it is not surprising that the QN method is faster than Newton's method, 13.7 seconds versus 16.2 seconds. Even though the low cost tridiagonal preconditioner is still being used, the number of solves has barely gone up so overall time is less. Notice that without the Watts line correction (the third method) the QN method does not do as well. This effect is also observed in Table 5.2 and Table 5.3.

Making the problem more difficult than the standard test case reveals more about the Quasi-Newton method in simulation. Doubling the oil and gas rate makes the problem much more difficult. Table 5.5 compares Newton, Quasi-Newton and modified-Newton. Observe that both of the latter methods have difficulties with the problem now. Each matrix solution method has required at least one time step cut. Indeed both methods suffer 2 time step cuts with the most powerful linear method tried.

The time step cuts were due to table look ups going out of bounds. This is not just due to the slower convergence rate of the latter two methods. The reason this happens is that by using an old Jacobian both methods lose some degree of implicitness; this becomes obvious when the problem becomes more difficult. Newton's method can in principle take any size time step on most any problem and usually converge but the same cannot be said about the last two schemes. The Quasi-Newton method does perform better than modified-Newton but both are unacceptable for this problem. If this last test represents the average difficulty of a simulation problem it is clear that reservoir simulation is too demanding for the Quasi-Newton method. However not all simulations will be as difficult since, for example, the test case itself is easier.

5.9.7. Performance on Approximate Jacobians

In black-oil simulation it is relatively simple to obtain an exact Jacobian, but this may not be the case in a compositional model. To simulate an approximate Jacobian, we perturb each 3×3 block of elements of the true Jacobian by alternately multiplying and dividing alternate blocks by a factor. Table 5.6 shows the results on these approximate Jacobians. All cases were solved to 0.01 tolerance so material balance error was negligible.

Observe that, as expected, the Quasi-Newton method does better as the approximation gets worse. This is in spite of using an accelerated scheme to solve the matrix. Also note that the Watts correction does not do as well with an approximate Jacobian. It seems that with an approximate Jacobian the correcting effect of the constraint may be nullified. This has been confirmed by other experiments.

Table 5.4. Depletion gas drive

	nIt	nF	nJ	nS	nW	Time sec	Oil Error	Gas Error
Newton								
zLine Jacobi	92	92	92	1258	0	87.5	1568	1633
red-black GS	60	60	60	701	0	61.6	720	746
RB GS + ORTH	43	43	43	150	0	19.0	96	107
RB GS+ORWatt	38	38	38	38	196	16.2	0	6
Quasi-Newton								
zLine Jacobi	61	20	20	765	0	45.5	208	221
red-black GS	51	16	16	439	0	34.8	192	213
RB GS + ORTH	50	16	16	169	0	19.0	64	63
RB GS+ORWatt	39	15	15	39	200	13.7	0	1

Table 5.5. Doubled oil and gas rates

	nIt	nF	nS	nW	Time sec	Oil Error	Gas Error
Newton							
zLine Jacobi	134	134	1662	0	111.6	9354	1598
red-black GS	81	81	752	0	67.0	389	136
RB GS + ORTH	65	65	190	0	26.5	61	10
RB GS+ORWatt	59	59	72	293	26.1	7	1
Quasi-Newton							
zLine Jacobi	145	47	1285	0	80.0	1c	
red-black GS	131	46	719	0	61.7	1c	
RB GS + ORTH	135	45	322	0	39.0	1c	
RB GS+ORWatt	156	44	164	588	47.2	2c	
modified-Newton							
zLine Jacobi	161	49	1448	0	91.8	1c	
red-black GS	135	46	882	0	70.8	1c	
RB GS + ORTH	247	69	446	0	55.1	1c	
RB GS+ORWatt	200	59	199	650	55.6	2c	

Table 5.6. Performance on approximate Jacobians

		nIt	nF	nS	nW	Time sec
Factor = 1.01						
Newton	rbzlGSO	48	48	203		53.6
	+Watts	47	47	79	377	53.8
QN	rbzlGSO	64	19	246		57.3
	+Watts	61	20	103	496	59.1
Factor = 1.025						
Newton	rbzlGSO	49	49	209		55.3
	+Watts	48	48	82	416	57.4
QN	rbzlGSO	66	20	253		59.6
	+Watts	61	20	105	509	61.0
Factor = 1.05						
Newton	rbzlGSO	61	61	285		73.7
	+Watts	66	66	111	690	84.3
QN	rbzlGSO	69	22	287		66.6
	+Watts	66	20	113	588	66.6
Factor = 1.075						
Newton	rbzlGSO	119	119	723		181.7
	+Watts	128	128	245	1685	182.1
QN	rbzlGSO	70	21	326		75.8
	+Watts	71	21	122	688	74.7

So the final picture on the Quasi-Newton method is mixed. It is clearly not as robust as Newton's method but for easy problems it can work faster than Newton's method. The ideal application seems to be in a simulator that uses a costly preconditioner but is being run on an easy problem. The Quasi-Newton method could thus be used as a backup option to be switched in when things are going well (e.g. after the initial transients have passed through the reservoir). With an approximate Jacobian the method becomes a strong contender for first choice. Another possible application for the QN method is if there a large number of equations at each grid block. Behie (1985) shows that the relative cost of factorization to backsubstitution varies from Neq to 5Neq depending on the type of preconditioner used.

5.10. SOLVING THE MATRIX EQUATIONS IN PARALLEL

The previous section has compared certain parallel matrix solution schemes with the two nonlinear methods. The details of the solution schemes are given now.

The first improvement over the Jacobi method is Red-Black Gauss-Seidel. This consists of coloring the grid in a checkerboard pattern of alternate red and black blocks. The red blocks are numbered first followed by the black blocks. For a 5×4 grid this gives the ordering shown in Fig. 5.10.

With the five-point finite difference scheme this ordering decouples each color's blocks since each of the 4 blocks around a red block are black and vice-versa. In this study for the 3-D simulations we color each *z-line* red or black and order the blocks by lines first and then by colors; this decouples each *z-line* from its surrounding lines. The corresponding matrix equation now becomes:

$$\begin{bmatrix} D_R & C_R \\ C_B & D_B \end{bmatrix} \begin{bmatrix} x_R \\ x_B \end{bmatrix} = \begin{bmatrix} b_R \\ b_B \end{bmatrix} \quad (5.3)$$

where D_R and D_B are both diagonal or block-diagonal, while C_R and C_B are of some unspecified structure. The red equation $D_R x_R = b_R$ can now be solved in parallel by solving any diagonal (or block-diagonal) element independently of the others. We solve each tridiagonal along a *z-line* independently of the others. This is followed by the solution of the black equation $D_B x_B = b_B - C_B x_R$ in the same manner. In first order iterative schemes the next iteration continues with $D_R x_R = b_R - C_R x_B$. When used as a preconditioner for Orthomin (Vinsome, 1976) just one pass through red and black is most efficient (i.e. C_R is ignored).

In the absence of round-off error and excepting pathological cases, Orthomin is guaranteed to solve a *n*th order matrix in at most *n* iterations by minimizing the residual in a series of orthogonal directions, provided space is available for 2*n* vectors. In practice only 2*k* vectors need be stored, *k* being between 5 and 10. Fig. 5.10 lists the procedure for Orthomin(*k*).

Most of the operations in Fig. 5.10 can be performed in parallel. The matrix-vector product in Step 4 can be done by multiplying by diagonals. This is simplest if the matrix *A* is in standard order. All vector operations can be done in parallel by allocating parts of the vectors among the processors.

If implemented as shown, Step 4 requires storing the matrix *A* in addition to the storage space for the LU factors of the preconditioner *M*. There are ways to avoid the extra storage, an

1	11	2	12	3
13	4	14	5	15
6	16	7	17	8
18	9	19	10	20

Figure 5.9. Red-black ordering of 5×4 grid.

To solve $Ax = b$, set $r = b$ and $x = 0$.

For $n = 0, 1, 2, \dots$ until convergence:

1. $l = n \bmod k + 1$
2. $m = \min(n, k)$
3. Solve $Mz = r$, where M is the preconditioning matrix, i.e. an easily solved approximation to A .
4. Form $u = Az$
5. For $i < l$ or $l < i \leq m$ Form $a_i = \gamma^i \langle p^i, u \rangle$
6. $q^l = z - \sum_{\substack{i=0 \\ i \neq l}}^n a_i q^i$
 $p^l = z - \sum_{\substack{i=0 \\ i \neq l}}^n a_i p^i$
7. $\gamma^l = 1 / \langle p^l, p^l \rangle$
8. $\omega = \gamma^l \langle r, p^l \rangle$
9. $x \leftarrow x + \omega q^l$
10. $r \leftarrow r - \omega p^l$

Figure 5.10. The Orthomin algorithm.

especially attractive method being the reduced system approach discussed by Behie and Forsyth (1984), which also halves the size of the matrix problem. It would be interesting to try this approach on a parallel machine but the ordering of the standard reduced matrix is not appropriate for parallel processing so some modification will be needed before it can be used.

Gustafson *et al.* (1988) avoid storing A and the matrix-vector product Az at the cost of an extra function evaluation, using

$$Az = \frac{b(x_0 + \epsilon z) - b(x_0)}{\epsilon} \quad (5.4)$$

This expression follows from the Taylor series approximation, $b(x + \delta x) = b(x) + A\delta x$, since A is the Jacobian of b .

For application in simulation the evaluation of the product this way has several disadvantages

- To satisfy the locally linear model represented by the Taylor series we must save the fluid properties of all three phases (or look them up again) at x_0 and use these to compute $b(x_0 + \epsilon z)$. Ordinarily storage for only one phase is needed so storage requirements are tripled if we wish to avoid another lookup.
- Any Gauss-Seidel type preconditioner part of the unfactored matrix A must be stored to compute the right hand side residual vector. In the Red Black ordering this is needed before the black pass through Eq. 5.3.
- Tests reveal that a straightforward multiplication by diagonals is 2.3 times faster than a standard function evaluation (i.e. one where the lookup is repeated).
- A second matrix-vector product with A is needed when constraints are used.

So the code multiplies by diagonals on Step 4; it turns out that the matrix multiply for the Watts line corrections is the most heavily used and could do with some fine tuning.

Step 3, the solution of the preconditioning matrix, is the crucial step in parallelization of the algorithm. The matrix M must be ordered so that it can be solved in parallel, hence the need for the various ordering schemes, red-black, domain decomposition, nested dissection etc.

As far as the QN method is concerned, Step 3 is the only place that holds promise for saving. The LU factors of the preconditioning matrix M are obtained once before the start of the Orthomin acceleration. In the QN method these factors can be retained for several non-linear steps during which the Jacobian is also not formed. This makes sense if obtaining the Jacobian and/or its factors is expensive and the number of Orthomin iterations per step is small.

5.10.1. Adding Constraints

The efficacy of constraints that force the residual to zero in a certain direction or over a set of equations has been demonstrated by Watts (1971), Settari and Aziz (1973), Appleyard *et al.* (1983) and Wallis (1983).

Watts (1971) proposed line correction to accelerate LSOR. Settari and Aziz (1973) proposed a 2-dimensional line correction technique. Appleyard *et al.* (1983) proposed the nested factorization method, which ensures zero residual sums along diagonal lines in two-dimensional problems and along diagonal planes in three-dimensional problems, as a preconditioner for Orthomin. Wallis (1983) proposed the use of residual constraints with Orthomin independently of the type of factorization used.

Wallis *et al.* (1985) proposed a new generalized conjugate residual method similar to Orthomin and proposed making the optimal choice of constraints based on the eigenvalues of AM^{-1} . They pointed out that the addition of constraints helps improve the robustness of preconditioners built with local grid information like ILU(0) or for that matter z-Line Jacobi or Gauss-Seidel. The constraints help to tie together preconditioners that use local information and eliminate low-frequency errors that otherwise eventually slow down the convergence.

In parallel computation we are, of necessity, using local information since we do not want one processor to depend on another. Killough and Wheeler (1987) used line corrections with a domain decomposition ordering and also noted that they provide a rigorous and simple technique to join the various solutions built with local information. While constraints are usually applied as line corrections, they may also be applied in other forms (for example to zero errors in pressure). In their most general form residual constraints require the solution of

$$C^T ACy = C^T r \quad (5.5)$$

where C is a constraint matrix of order $n \times m$, A is of order $n \times n$, and m is small compared to n . This correction satisfies the constraint condition

$$C^T r_c = 0 \quad (5.6)$$

where

$$r_c = r - ACy \quad (5.7)$$

The constraint condition forces the components of the corrected residual r_c to sum to zero in the directions chosen by C .

The solution y can be added to the current solution or the residual can be updated by using Eq. 5.7. The choice of C defines the direction in which the constraints are applied. Generally the corrections should be applied in the direction of largest transmissibility (Aziz and Settari, 1979) and this is almost always the vertical direction. Wallis *et al.* (1983) used eigenvalue analysis to determine the optimal direction for the corrections and these invariably turned out to be in the direction of the highest transmissibility. Killough and Wheeler (1987) looked at a heterogeneous problem with no dominant direction for the transmissibility and observed that applying corrections simultaneously in all three directions was best in such cases.

For this study we have only looked at the use of constraints with Red-Black orderings. The Red-Black ordering is highly scalable since there the number of blocks of a given color can be tailored to fit the number of processors, furthermore the size of the submatrix problems does not change as it does in a domain decomposition ordering. To the Red-Black ordering we add z-line corrections since the vertical direction usually has the highest transmissibility and also because the primary ordering is along z-lines.

5.10.2. Solving Matrix and Constraint Equations in Parallel

The constraints can be added to the Orthomin acceleration scheme by Wallis's (1983) CRPO method - Constrained Residual Preconditioning for Orthomin. This is very similar to Orthomin and is shown in Fig. 5.11.

There are now two matrices: A , the original $n \times n$ matrix, and C^TAC , the $m \times m$ constraint matrix, to be solved in parallel. Using z-line corrections is convenient for the constraint matrix retains the same structure as the original matrix with three-dimensional features replaced by two-dimensional features. For Red-Black z-line ordering of the original matrix, the constraint matrix is Red-Black point ordered and can be solved in parallel in exactly the same manner as the original matrix.

Acceleration is also used to speed up convergence on the constraint equations. The outer Orthomin loop of Fig. 5.11 now has an inner Orthomin loop for the correction at Step 4. As an inner loop the corrections tend to be expensive. So even though they greatly improve convergence of the outer iterations [see the huge drops in (nS) in Tables 5.2 - 5.5] the number of inner iterations is large (nW) and total time may decrease only slightly.

Table 5.7 the effect of residual constraints on linear convergence and total time. For loose linear tolerances the benefit of the constraints is not clear, but for tight linear tolerances an appreciable benefit of the constraints appears. Also note that in every case there seems to be an optimum on the Watts tolerance on either side of which total time increases. With tight tolerance, the number of solve steps goes down quite a bit but the number of Watts iterations increases to counter the benefit. There is the possibility of another level of correction on top of the Watts level to reduce the Watts time. From the Table 5.7, we can obtain some guidelines for practical use of the constraints.

- The Watts tolerance has to be fairly loose in order to keep the time spent on corrections within reasonable bounds.
- The Watts program code should be highly optimized. One unfortunate consequence of the reduced size of the Watts matrix is that the granularity for parallel computation is much reduced so speedup is also affected.
- The constraints become more useful as the problem becomes more difficult but costs also go up.

Later runs show that the constraints become more effective as the areal extent of the problem increases. Table 5.8 shows the effect of varying the number of blocks in the z direction on residual constraints. With up to 10 grid blocks in the vertical direction the effectiveness of the constraints increases dramatically. However the effectiveness falls off once the number goes beyond that; this may be because the length of the z-line is then greater than the length in either of the other two directions.

5.11. EXPERIMENTS WITH THE PARALLEL SIMULATOR

With the fully parallel simulator at hand and the Newton method used as the primary method, runs were made on the simulator to experiment with its abilities.

Table 5.7. Effect of residual constraints on linear convergence.

	Watts				Time
	Tol	nIt	nSolve	nWatts	sec
Linear Tolerance = 0.1					
W/o Correction	-	57	139	0	43.0
W Correction	0.25	50	53	232	43.3
W Correction	0.10	48	51	303	45.5
Linear Tolerance = 0.01					
W/o Correction	-	44	189	0	48.6
W Correction	0.25	46	75	255	47.0
W Correction	0.10	43	68	345	47.5
Linear Tolerance = 0.001					
W/o Correction	-	43	304	0	71.4
W Correction	0.25	43	112	337	55.3
W Correction	0.10	43	102	468	58.8
W Correction	0.05	43	92	527	59.5
Linear Tolerance = 0.0001					
W/o Correction	-	43	398	0	91.8
W Correction	0.25	43	150	451	67.9
W Correction	0.10	43	125	564	67.2
W Correction	0.05	43	113	652	69.0
Linear Tolerance = 0.00001					
W/o Correction	-	43	485	0	111.1
W Correction	0.25	43	197	574	83.1
W Correction	0.10	43	154	684	78.1
W Correction	0.05	43	139	778	79.9

Table 5.8. Effect of z-line length on residual constraints.

	Watts				Time sec
	Tol	nIt	nSolve	nWatts	
			10 x 10 x 3		
W/o Correction	-	44	209	0	23.6
W Correction	0.2	44	75	287	22.7
			10 x 10 x 5		
W/o Correction	-	46	216	0	44.0
W Correction	0.2	45	73	295	36.6
			10 x 10 x 10		
W/o Correction	-	46	184	0	81.5
W Correction	0.2	44	76	273	69.5
			10 x 10 x 15		
W/o Correction	-	45	172	0	115.2
W Correction	0.2	45	84	289	110.0
			10 x 10 x 20		
W/o Correction	-	47	173	0	157.7
W Correction	0.2	45	94	327	150.3
W Correction	0.1	45	94	414	152.3

To solve $\mathbf{Ax} = \mathbf{b}$, define $\mathbf{T} = \mathbf{C}(\mathbf{C}^T \mathbf{A} \mathbf{C})^{-1} \mathbf{C}^T$, and compute $\mathbf{x} = \mathbf{Tb}$. Set $\mathbf{r} = \mathbf{b} - \mathbf{Ax}$. For $n = 0, 1, 2 \dots$ until convergence:

1. $l = n \bmod k + 1$
2. $m = \min(n, k)$
3. Solve $\mathbf{Mz} = \mathbf{r}$, where \mathbf{M} is the preconditioning matrix of \mathbf{A} .
4. Solve for corrections and add to solution $\mathbf{z} \leftarrow \mathbf{z} + \mathbf{T}(\mathbf{r} - \mathbf{Az})$
5. Form $\mathbf{u} = \mathbf{Az}$
6. For $i < l$ or $l < i \leq m$ Form $\mathbf{a}_i = \gamma^i \langle \mathbf{p}^i, \mathbf{u} \rangle$
7. $\mathbf{q}^l = \mathbf{z} - \sum_{\substack{i=0 \\ i \neq l}}^n \mathbf{a}_i \mathbf{q}^i$
 $\mathbf{p}^l = \mathbf{z} - \sum_{\substack{i=0 \\ i \neq l}}^n \mathbf{a}_i \mathbf{p}^i$
8. $\gamma^l = 1 / \langle \mathbf{p}^l, \mathbf{p}^l \rangle$
9. $\omega = \gamma^l \langle \mathbf{r}, \mathbf{p}^l \rangle$
10. $\mathbf{x} \leftarrow \mathbf{x} + \omega \mathbf{q}^l$
11. $\mathbf{r} \leftarrow \mathbf{r} - \omega \mathbf{p}^l$

Figure 5.11. Constrained residual preconditioning for Orthomin.

5.11.1. Comparison of Parallel Algorithms

Table 5.9 compares some parallel algorithms on a serial machine. The first is point-Jacobi Orthomin which is similar to the method used by Gustafson et al. (1988). Note that we always solve the three equations at each grid block simultaneously, so the point qualifier refers to the number of grid blocks involved.

It is clear that for reservoir simulation a point-Jacobi preconditioner is unacceptable. Time is an order of magnitude worse and, more seriously, the material balance is badly flawed. So in spite of the great speedup shown with this preconditioner it is not going to work for simulation.

The other three methods turn in more respectable times. The effectiveness of Red-Black Gauss-Seidel is obvious when compared to the equivalent Jacobi method. The Watts corrected version shows a slight edge on this 100 day simulation run.

One of the questions that arise in evaluating the parallel speedup of an algorithm is how it compares with the best serial algorithm. The test case simulation was run to 10 years on IMEX, a commercial simulator, returning a time of 198 seconds. On the same simulation and the same serial machine the Red-Black zLine Gauss-Seidel method took 418 seconds, while the Watts corrected version took 332 seconds (both had linear solution tolerance of 0.01, and zero material balance error). A further run with 0.1 linear tolerance returned a time of 257 seconds and very good material balance.

The run times have clearly suffered because of using a parallel scheme. However they are within the same range of magnitude; there is no need to make the order of magnitude sacrifice entailed with the point-Jacobi scheme as shown in Table 5.9.

Part of the reason the runs are slower is that the simulator is research code built from scratch and does not have optimized code and algorithms everywhere while IMEX is a state-of-the-art commercial simulator. Time step choice also plays a factor. It is likely that with more powerful algorithms this gap can be narrowed. For example, the z-line preconditioner can be replaced by a block preconditioner to solve for all the grid blocks residing on a processor. Also different ordering schemes could be tried. The main conclusion to draw is that it will be possible to solve in a parallel manner without sacrificing a great deal of performance in serial comparison. Secondly the Watts' correction can help improve the competitiveness of the Red-Black scheme especially over long and difficult runs. Once acceptable serial times are obtained, running in parallel can make a big difference, for example with parallel speedup of 10, turnaround time for the Watts corrected version drops to 33.2 sec.

Figure 5.12 shows the parallel speedup obtained with varying problem sizes on seven time steps of the test problem. The maximum speedup achieved is 10.46 on 14 processors. The kinks in the curves are caused by variations in load balance. For example the $12 \times 12 \times 3$ problem is perfectly balanced on 12 processors and a peak in speedup reflects this. On the other hand the $10 \times 10 \times 3$ problem suffers from load imbalance at 12 processors since the grid blocks cannot be equally divided among the processors. In practice the grid blocks are assigned one z-line at a time, so all operations along a z-line are handled by the same processor. This makes for fairly coarse granularity so quite a bit of time is wasted by processors that are one z-line short, hence the pronounced peaks and dips.

Figure 5.13 shows better speedup on this 3-phase three-dimensional problem compared to the 2-phase two-dimensional problem tried earlier. The positive slope is once again

Table 5.9. Comparison of some parallel algorithms.

nIt	nSol	nWatts	Time	Oil Error	Gas Error
Jacobi-Orthomin					
100	2392		275.5	34366	8566
zLine Jacobi-Orthomin					
51	404		35.1	16	6
Red-Black zLine Gauss-Seidel-Orthomin					
44	210		23.4	16	4
Red-Black zLine GS-Orthomin + Watts-Orthomin					
44	75	287	22.2	0	2
CPU time for 10 yr simulation run					
IMEX				198	seconds
Red-Black zLine GS				418	seconds
RB zLine GS-Watts				332	seconds

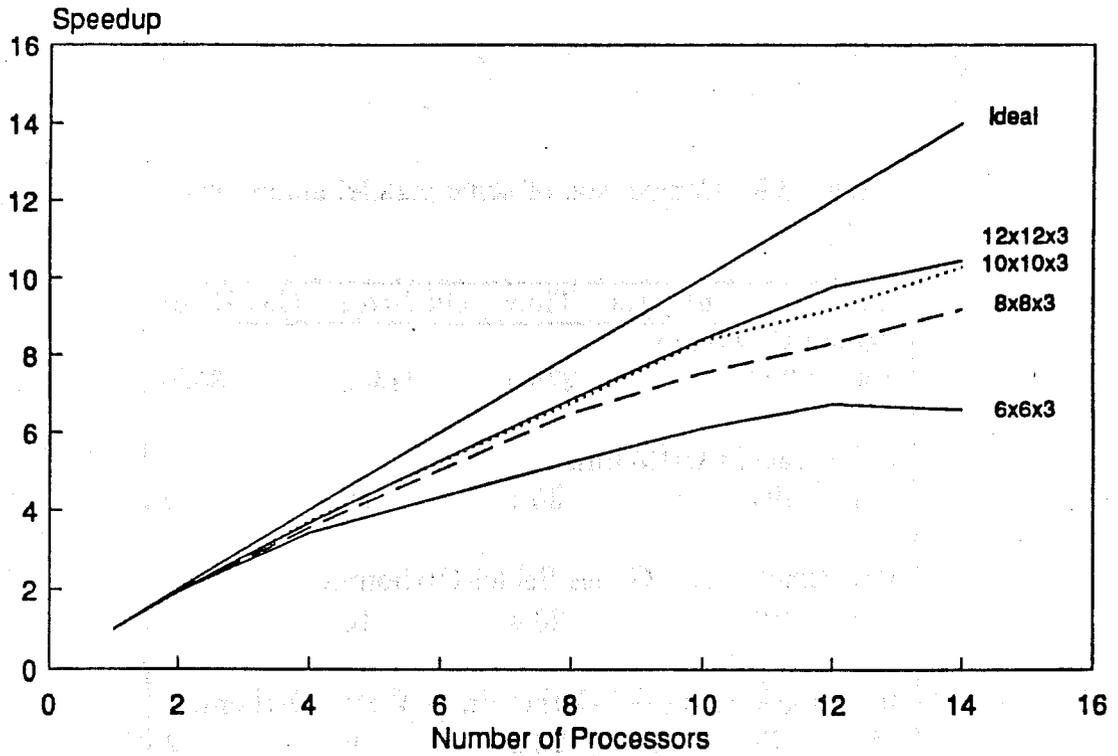


Figure 5.12. Effect of problem size.

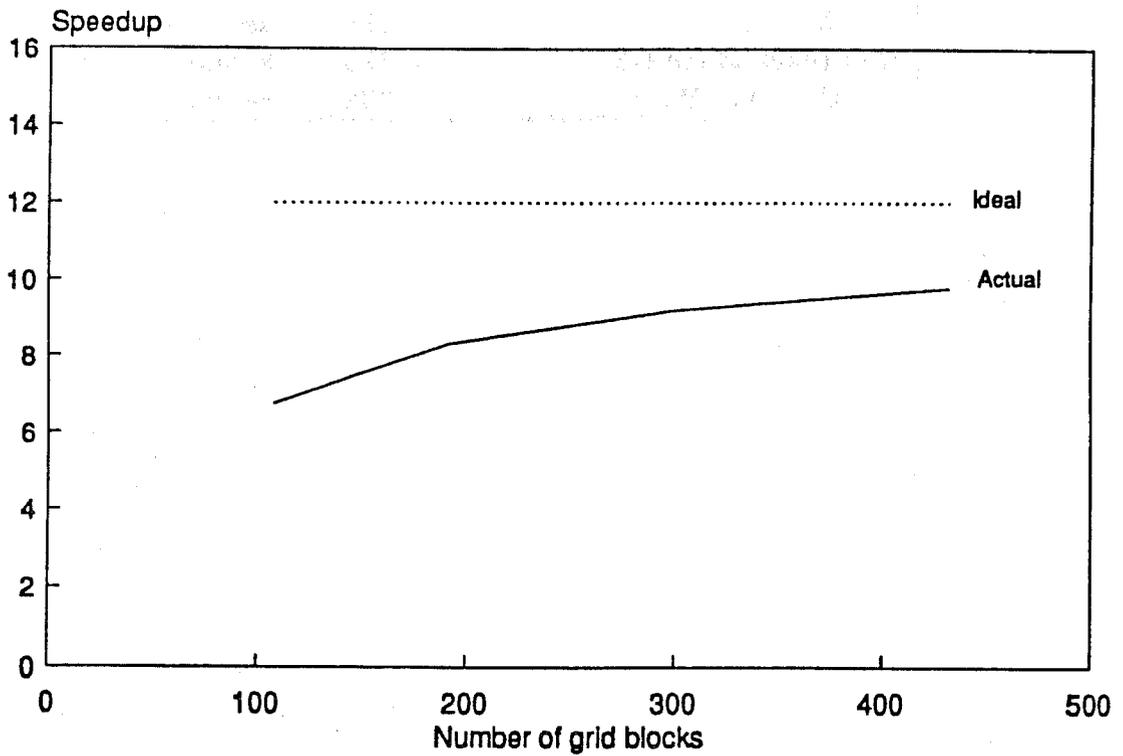


Figure 5.13. Speedup on 12 processors.

encouraging. However, Fig. 5.14 shows that on 14 processors things are not as good. Part of the reason is the imperfect load balance with this odd number of processors. Part of this is also due to the effect noted earlier - increasing the number of processors makes it very hard to get good speedup - the acclaim accorded to Gustafson *et al.* (1983) is obviously very well deserved. On the other hand, the largest problem run here allocates only 31 grid blocks per processor when 14 are used in parallel; bigger problem sizes could not be run due to memory limits on the machine.

5.11.2. Effect of Constraints on Speedup and Total Time

Figure 5.15 shows how the Watts line correction affects parallel speedup on the $10 \times 10 \times 3$ problem. The small size of the constraint matrix means that the processors are unable to do much work before synchronization is needed. As the number of processors increases, the amount of work per processor decreases and the problem gets worse. Relaxing the linear tolerance from 0.1 to 0.2 relieves some of the slowdown. The problem is more obvious on the $6 \times 6 \times 3$ problem which is shown in Figure 5.16. Part of the effects seen are because of a peak at the well-balanced 12 processor run, but here even the 0.2 correction causes a big drop.

Figures 5.15 and 5.16 showed *speedup* i.e., the time for a problem on several processors compared to the time for the same problem on one processor. However the effectiveness of the Watts line correction depends on the problem as shown earlier in Table 5.7. Fortunately the effectiveness increases with problem size. Figure 5.17 shows the ratio of *total* time with Watts line correction to the time without correction, on a single processor. Evidently the correction becomes more useful as the problem size goes up; its function of tying together the local solutions becomes more important as the solutions move far apart.

Because of the increased effectiveness with problem size shown in Fig. 5.17 and the increased size of the constraint matrix, the total parallel time with corrections eventually becomes competitive as shown in Fig. 5.18.

5.11.3. Effect of Input/Output on Speedup

Figure 5.19 illustrates how even small serial components affect parallel speedup. At the end of every time step, the simulator on processor 0 normally prints out five lines of information describing details of the last time step. This information is calculated beforehand and this figure shows the effect of simply including or suppressing printout of these lines. Observe that even this small serial component affects speedup noticeably. On a production simulator it will probably be necessary to use parallel I/O and moreover a good deal of effort will probably be needed in going through all operations to eliminate small serial inefficiencies. The problem is less severe with a small number of processors, observe that on four processors the speedup is barely affected.

5.11.4. Speedup of Various Tasks in Simulation

One of the objectives of this study was to build a fully parallel simulator with all operations done in parallel except unavoidable serial components. To see how much speedup had been achieved, a series of runs were made to time the various tasks running in parallel.

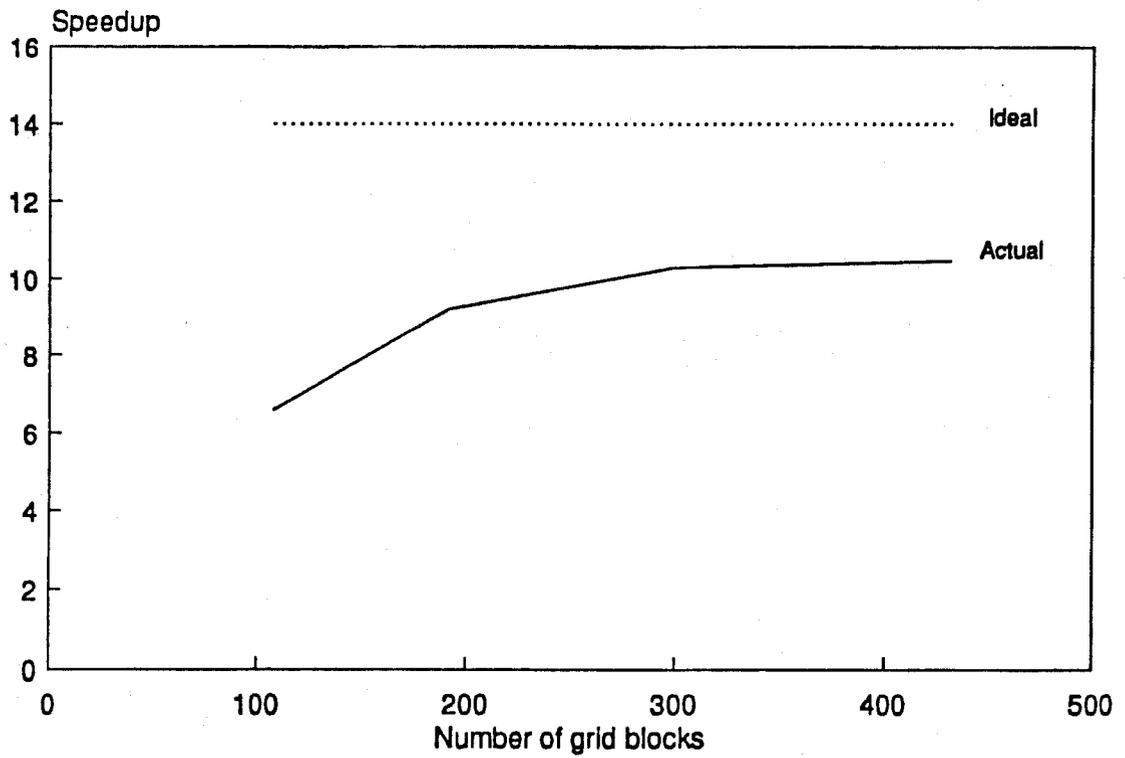


Figure 5.14. Speedup on 14 processors.

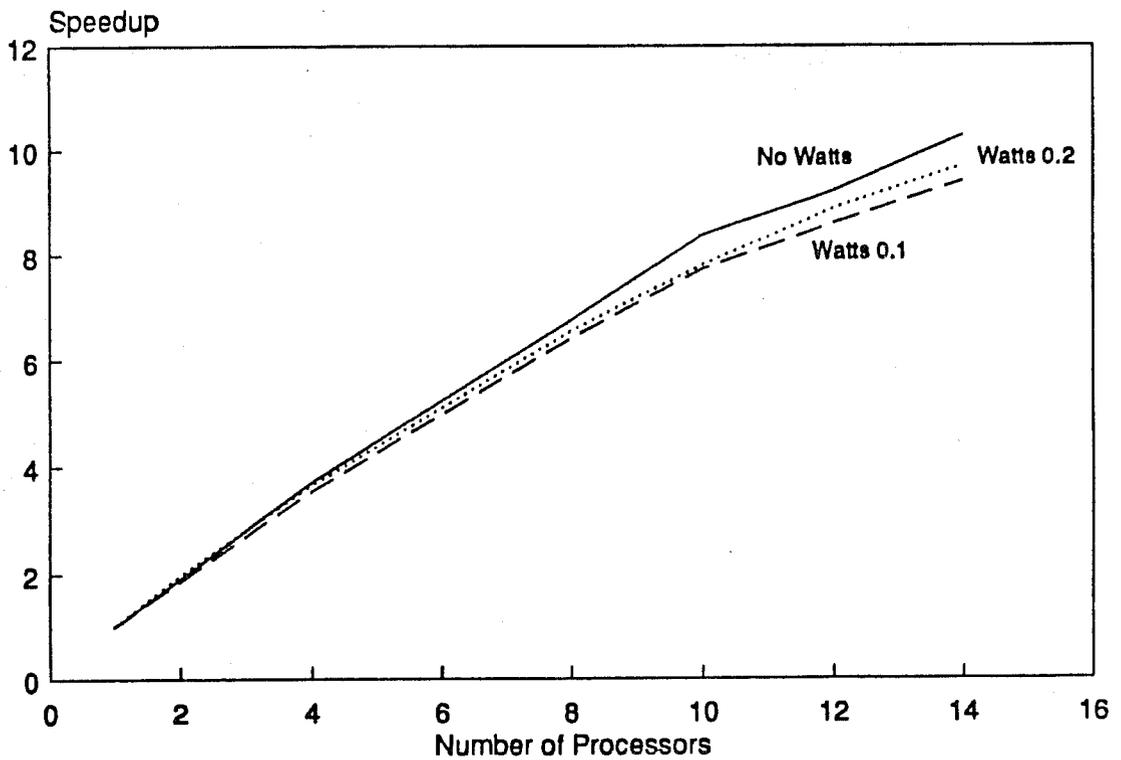


Figure 5.15. Effect of corrections on speedup on $10 \times 10 \times 3$ problem.

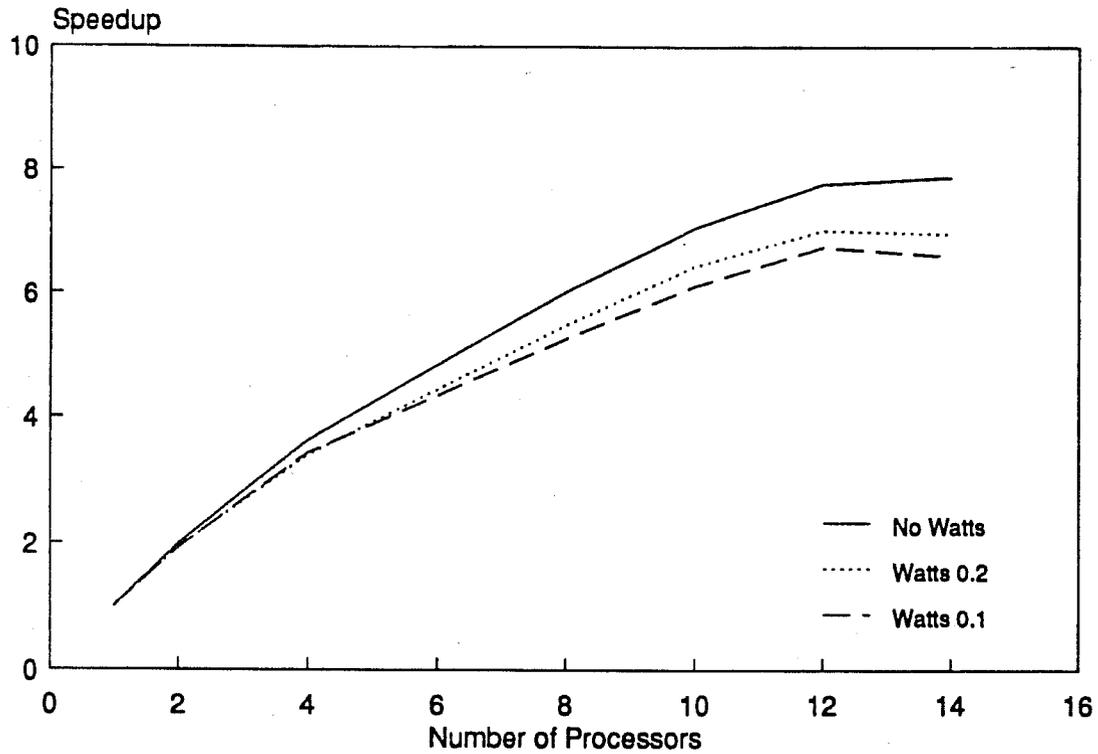


Figure 5.16. Effect of corrections on speedup on $6 \times 6 \times 3$ problem.

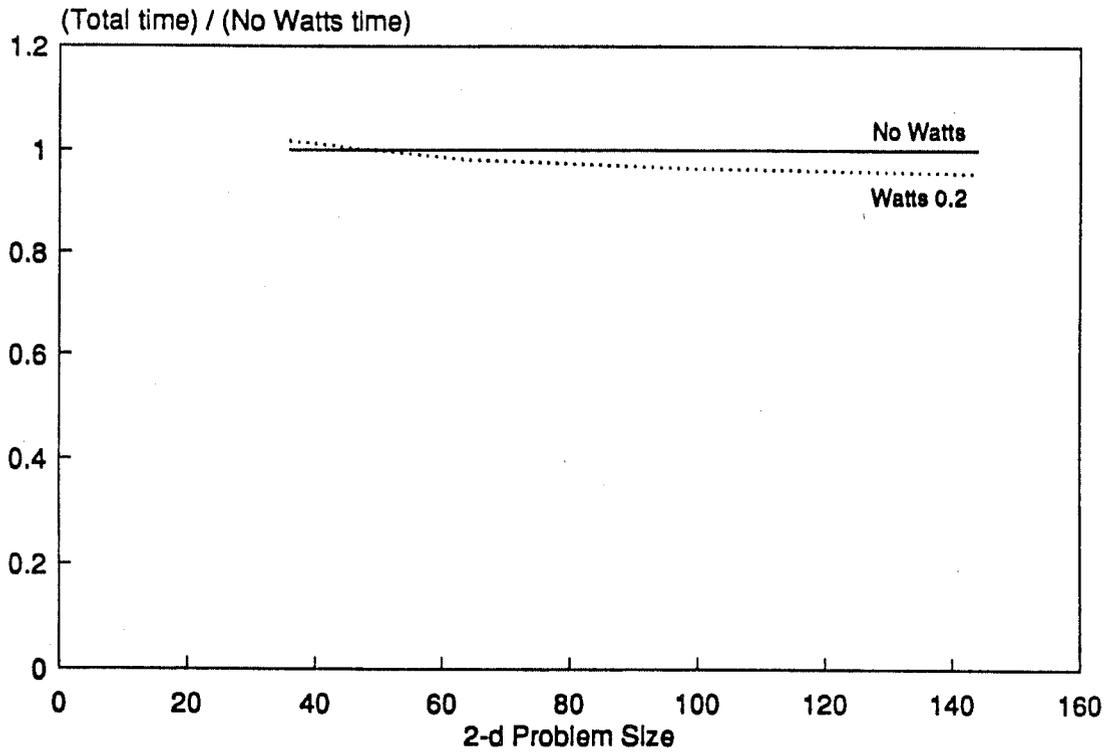


Figure 5.17. Effect of corrections on total time on 1 processor.

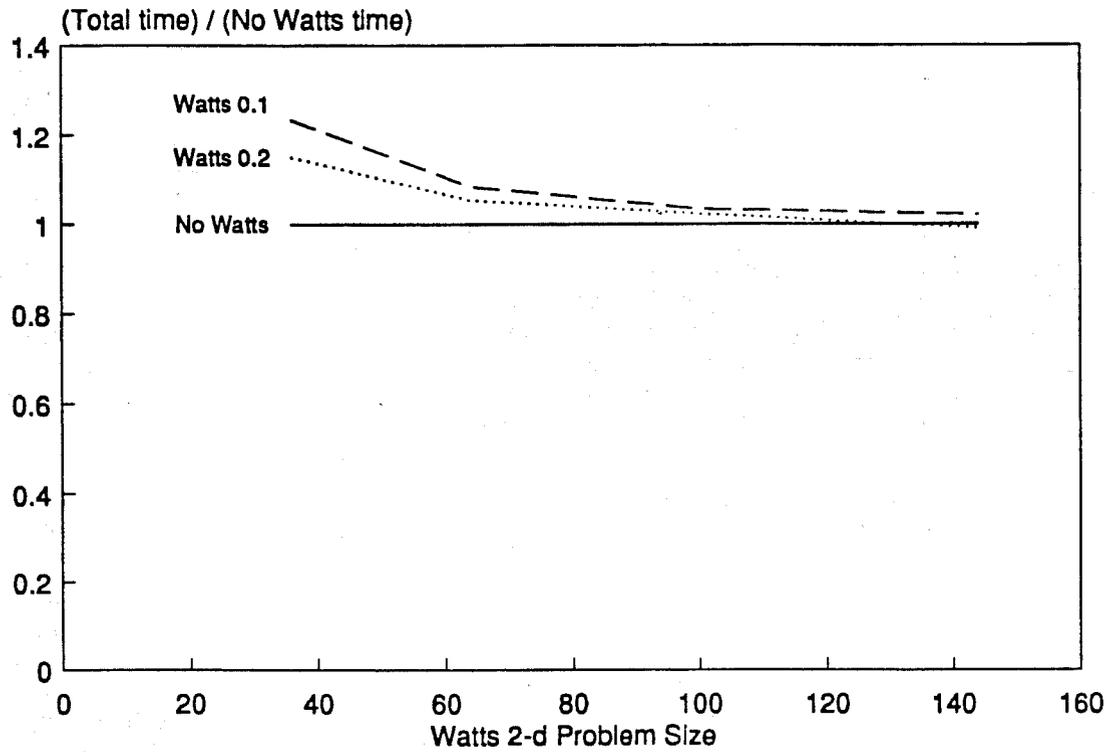


Figure 5.18. Effect of corrections on total time on 14 processors.

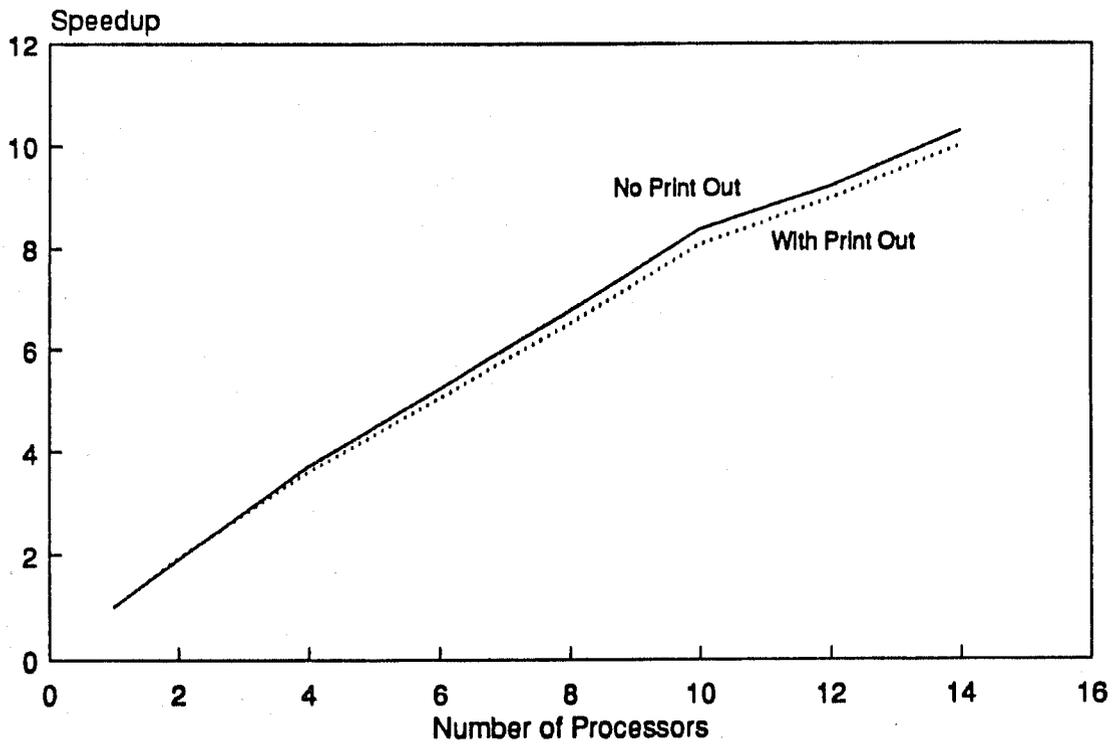


Figure 5.19. Effect of I/O on parallel speedup.

The amount of serial time spent on the tasks depends on the time step, as shown in Fig. 5.20 for time steps of 1 day and 100 days. The cost to form and factorize the Jacobian is, of course, unchanged in absolute time. However relative to the total computation time the cost to form the Jacobian changes from 24% to 18% with time step size. The factorization cost similarly changes from 32% to 24%, (this includes the cost of forming and factorizing the Watts matrix). For both time step sizes the solve step is the dominant expense (for this preconditioner and this test problem). This includes all steps needed to get a solution vector i.e. forward-elimination, back-substitution, acceleration and the solve step for the Watts matrix.

Figure 5.21 shows the speedup of various solve options on a one day time step. At this small time step size the first order Red-Black Gauss-Seidel is competitive and in fact has the highest speedup. This avoids the several dot-products and small granularity vector operations in Orthomin so speedup obtained is good; the peak speedup is 11.29 which is the best speedup obtained for any operation in this study. As the solve step increases in power with the addition of acceleration and line corrections, speedup decreases progressively.

On the 100 day time step, acceleration must be used to get acceptable performance. Figure 5.22 shows that the speedup obtained here without line corrections remains virtually the same as that obtained with a one day time step. The speedup obtained with line corrections is hampered because of the greater time spent on the Watts matrix to handle this difficult problem, compared to that spent in the one day time step problem (in both cases the Watts problem is solved to 0.1 tolerance). At the bottom of this figure is a comparison of the total time without corrections relative to time with corrections. With one processor the corrected version is more efficient by a factor of 20%. However due to its decrease in efficiency in parallel the corrected version is eventually less efficient by 2% at 14 processors. In spite of this small loss in efficiency in parallel it is likely that once the problem size gets large enough, the corrected version will be preferable always. Also this big drop in speedup suggests that the Watts correction is a critical area in which to seek small gains in efficiency. For this problem the Red-Black point-Gauss-Seidel used on the constraint matrix turns out to be more efficient than an x-line or y-line Jacobi so there is no obvious parallel method to employ for increased efficiency.

Figure 5.23 shows the speedup for forming and factorizing the Jacobian, including factorizing the Watts matrix. Although both are results on a $12 \times 12 \times 3$ problem, it can be seen that the speedup behavior for forming the Jacobian is a little erratic. It is also apparent that forming the Jacobian is less efficient than factorizing it. Part of this is due to the relatively small granularity of the various tasks in forming the Jacobian - when all BARRIERS were removed maximum speedup rose to 10.93. Part of the slowdown is that because of the presence/absence of wells and reservoir boundaries the load is not perfectly balanced when forming the Jacobian.

In principle the factorization should run at nearly ideal speedup, since the steps are completely independent. A sample code for the factorization is shown in Fig. 5.24. This code is run by each processor. The function `b+IsMyLine+` is used to check if the *i*th z-line is the responsibility of the processor and just requires two integer comparisons to return TRUE or FALSE.

In spite of this nearly ideal parallel code, the speedup is not close to ideal, although BARRIERS are very few and far-between. Contributing to the loss of ideal speedup are issues concerned with the basic hardware of the system. In particular the MULTIMAX is built with two processors on a card sharing the same cache. With two processors on the cache, contention for the cache will take place - hence the machine is not fully scalable. Another factor that may

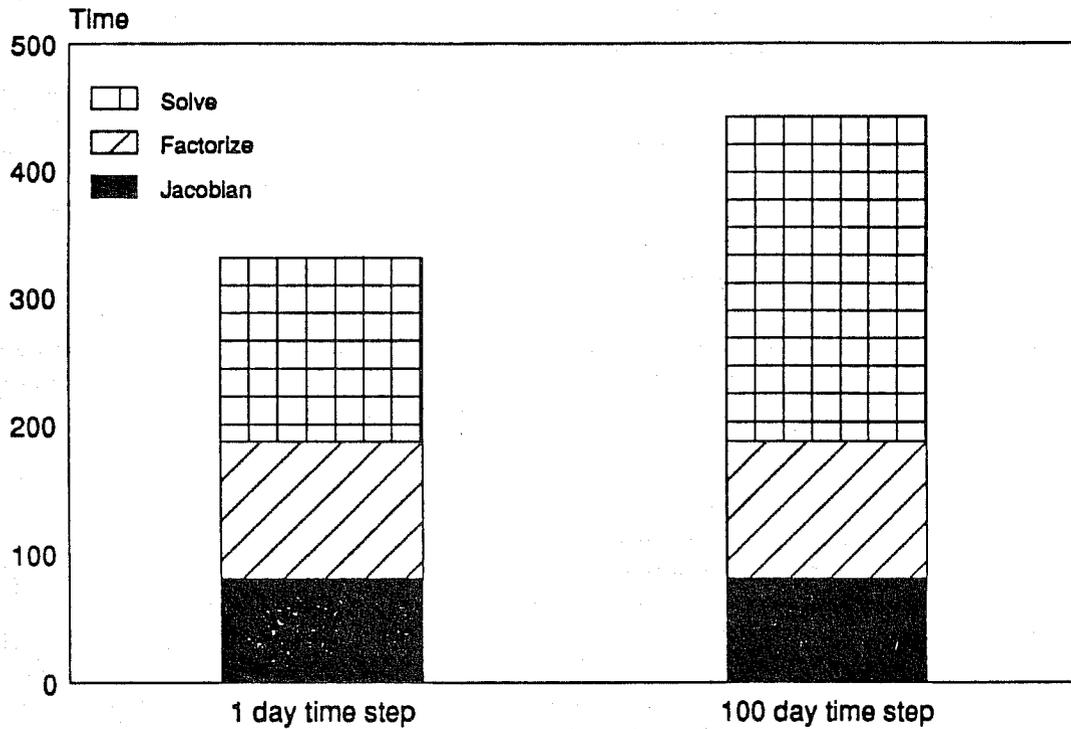


Figure 5.20. Time spent on the major tasks in simulating the test running in parallel.

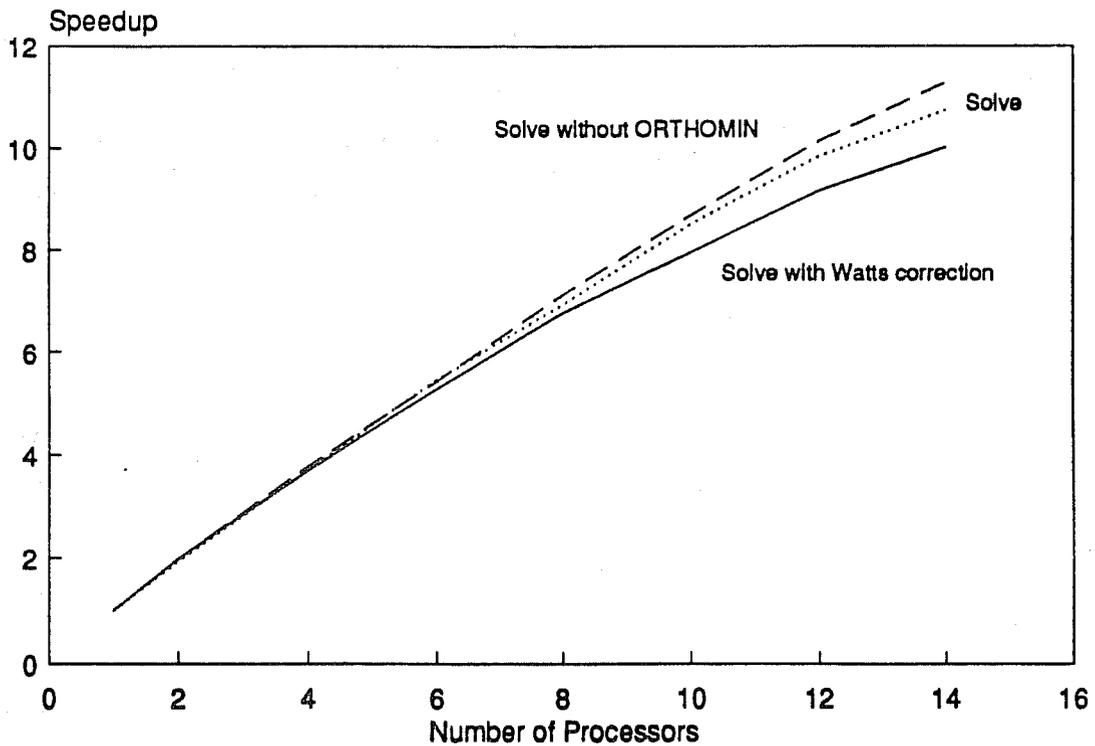


Figure 5.21. Speedup of solve step for 1 day time step.

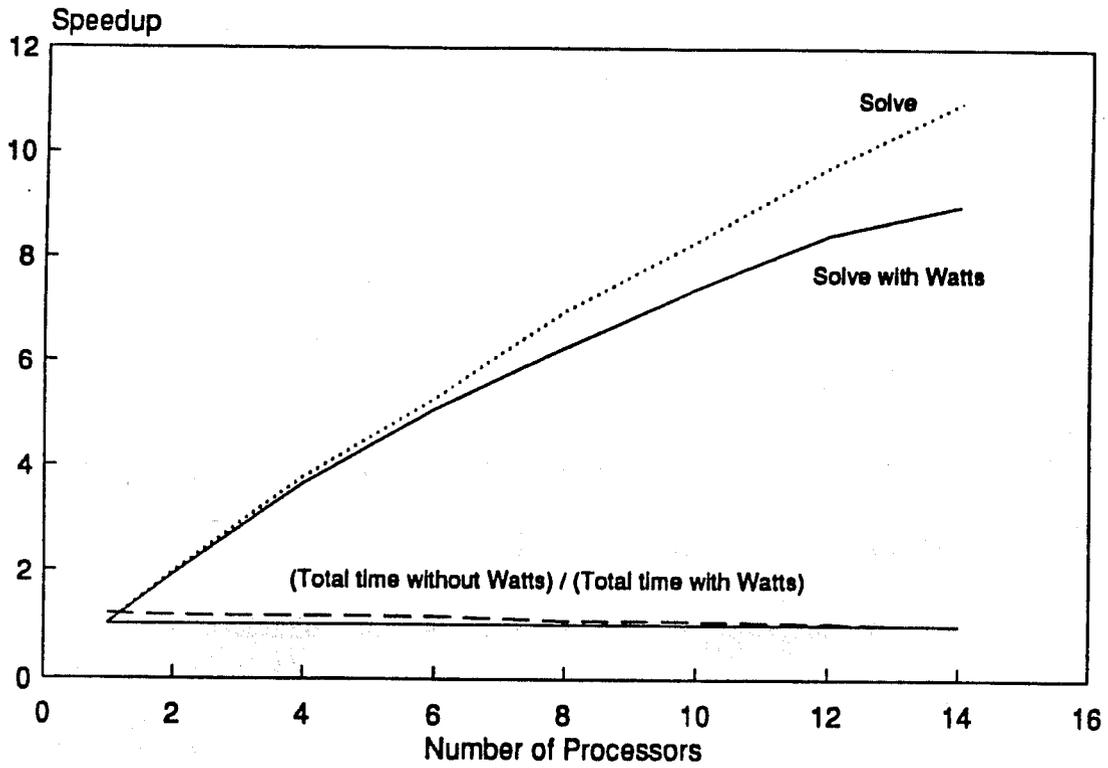


Figure 5.22. Speedup of solve step for 100 day time step.

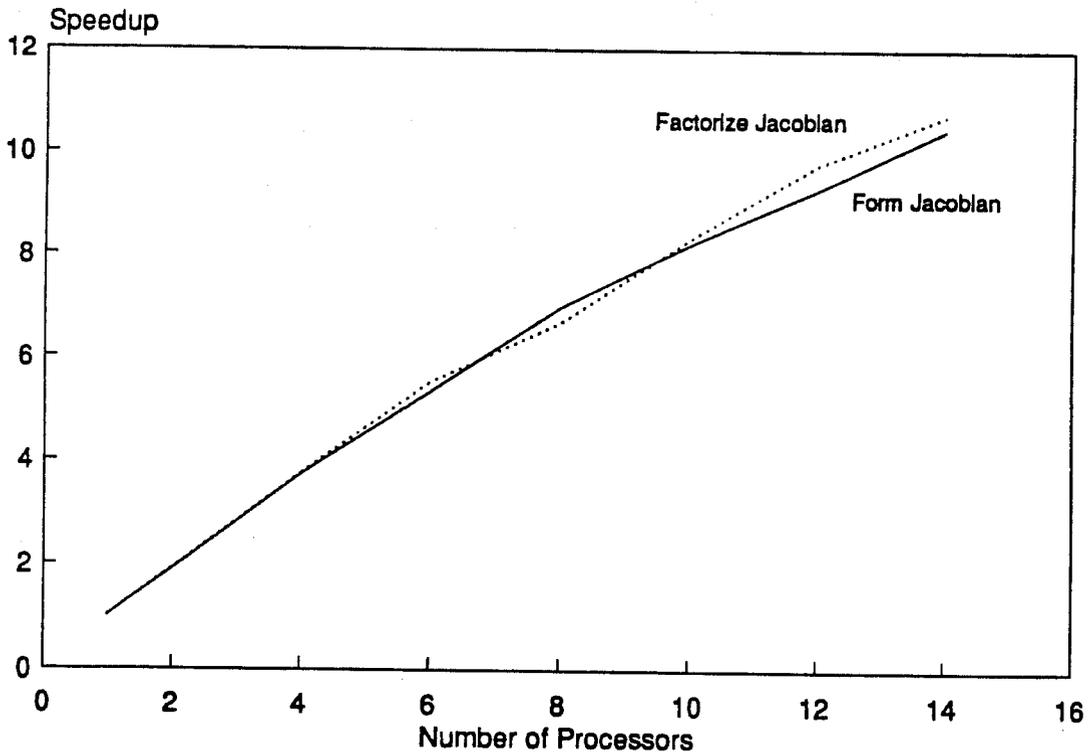


Figure 5.23. Speedup of forming and factorizing Jacobian.

```
BARRIER();                /* wait for Matrix to be ready */

CopyDiagonalsToSpaceForLfactors();

    /* Lower triangular needs to be saved for black pass */
BARRIER();                /* Wait till done */
for (i = 0; i < nIJ; i++) {
    if (IsMyLine(i)) {      /* see if this line is your job */
FactorizLine(i);
MakeAndFactor1WattsPoint(i);
    }
}
BARRIER();
```

Figure 5.24. Example code for factorizing by lines.

be contributing to the loss of speedup is cache line size. Commonly if any cache data is invalid - just the affected bytes are not the only ones replaced - typically an entire line of bytes (16-32) will be replaced. Now it is possible that part of a line is affected by one processor, while part of is affected by a second processor - this may cause loss of speedup as lines are replaced successively on each processor's write.

6. CONCLUSIONS

This study has compared the advantages and disadvantages of various Newton related methods in well testing, optimization and reservoir simulation. The study has also looked at the question of optimizing the production and injection schedules of the cyclic steam injection process, and finally looked at parallel reservoir simulation. The comparison of Newton type methods has shown the following:

- Second order methods in automated well test analysis tend to be unaffected by the introduction of ill-defined parameters, while the (first-order) Gauss-Marquardt method is seriously affected. However unless initial guesses are good the second-order methods do not usually do as well as first order methods.
- By replacing small and negative eigenvalues of the Hessian with large positive constants, we can improve the robustness of the otherwise fragile Newton method.
- In optimization the Quasi-Newton method converges rapidly and saves on function evaluations over Newton's method.
- In simulation the Quasi-Newton method may in certain cases be used as an alternative to Newton's method. The ideal application for it seems to be when a high cost preconditioner is used on a relatively easy problem.
- When exact Jacobians are available its application should be as a backup for a different primary method.
- With approximate Jacobians the Quasi-Newton method may outperform Newton's method in speed of convergence and total time.
- The Quasi-Newton method seems to work better when the Watts line correction is used to damp out low frequency errors.
- If direct solution techniques are being used (as is often the case in research type programs) the Quasi-Newton method can save significant amounts of time compared to Newton's method.

Experiments in optimizing the cyclic steam injection process suggest certain improvements for optimal operations:

- Soak time should be minimized or eliminated.
- Steam injection volumes considerably larger than current field practice may improve production rates by keeping the well flowing strongly for a longer period thereafter.
- Optimizing a combination of parameters instead of just one leads to a more desirable set of operating conditions.
- A high steam injection temperature allows greater possibility of improvements than a high steam quality because average temperature stays high.

Constructing a scalable parallel simulator has revealed the following guidelines and conclusions:

- It will be possible to build a fully scalable simulator and obtain serial times that are fairly close to the best serial algorithm. However due to slowdowns at BARRIERS etc. the achieved speedup in parallel is highly dependent on problem size. Speedups of 10.5 on 14 processors were attained on the relatively small problems tested.
- For parallel decomposition the program should be structured so that each task is completed in parallel at the same time and overlap of tasks should not be used. All processors then work on the same routine at the same time.
- Programming to the message-passing paradigm may be beneficial even on a shared-memory machine because:
 - The number of BARRIER statements can be reduced substantially.
 - Local memory is faster than global memory and by using this paradigm all arrays and vectors can then be stored in local memory.
 - We avoid cache and memory contention by localizing a processor's work to one region of memory.
 - The code is more easily ported to a message-passing machine.
- Residual constraints can improve the robustness of parallel preconditioners especially as the simulation problem gets harder.
- As the simulation problem gets harder more time is spent on the constraint matrix. Because of the small size of the constraint matrix parallel speedup drops off as the problem get harder.
- Efficient solution of the constraint matrix appears to be a critical step towards obtaining the best performance.

NOMENCLATURE

A	Matrix
A_k	Approximate Jacobian matrix
b	Vector
b_k	Temporary vector used by Quasi-Newton method
B_l	Formation volume factor of phase l, res. bbl/STB
c_k	Quasi-Newton update vector
C	Constraint matrix
d	Temporary solution for Quasi-Newton method
e_i	Unit vector with non-zero ith element
f	Nonlinear system of equations
f_{HD}	Dimensionless heat loss from horizontal faces of heated zone
f_{PD}	Dimensionless heat loss with produced fluids
f_{VD}	Dimensionless heat loss from vertical faces of heated zone
F	Nonlinear function (scalar-valued)
g	Gradient vector of F
h	Step used for finite-difference derivatives
h_t	Formation thickness
H	Hessian matrix of F
H_{inj}	Heat injected in current cycle
H_{last}	Heat remaining in reservoir from previous cycles
J	Jacobian matrix of f
k	Absolute permeability
k_{rl}	Relative permeability of phase l
K_R	Reservoir conductivity
m_l	History match parameter
M_l	Heat capacity of phase l
M	Preconditioning matrix
p	Step vector
p_l	Pressure in phase l
P	Parallelizable fraction of time spent on one processor
P_{cow}	Oil-water capillary pressure
P_{cog}	Oil-gas capillary pressure
q	A-orthogonal step vector
q_l	Flow rate of phase l (per unit volume in simulation)
Q_l	Total flow rate of phase l from gridblock
Q_{max}	Heat in heated zone before start of production
Q_p	Heat lost with produced fluids
r	Residual vector
r_w	Wellbore radius
R_h	Heated zone radius

R_s	Solution gas-oil ratio SCF/STB
R_x	Radial distance along cone
s	Step vector
S	Serial fraction of time spent on one processor
S_1	Saturation of phase 1
t	Time
t_{inj}	Injection time
t_{soak}	Soak time
t_{HD}	Dimensionless time for heat loss from horizontal faces of heated zone
t_{VD}	Dimensionless time for heat loss from vertical faces of heated zone
T_{avg}	Average heated zone temperature
T_1	Transmissibility of phase 1
T_p	Time to execute program on p processors
T_R	Reservoir temperature prior to steam injection
T_S	Steam injection temperature
T_1	Time to execute program on one processor (assumed to be unity)
V	Block volume
x	Distance in x direction
x	Vector of unknowns
y	Yield, i.e. change in function value after nonlinear step
z	Depth from reference point

Greek Letters

α	Thermal diffusivity
γ	Lipschitz constant
γ_1	Density of phase 1
λ_1	Mobility of phase 1
μ_1	Viscosity of phase 1
ρ	Step length parameter
ϕ	Porosity
Phi	Potential

Subscripts

g	Gas
i	Gridblock position in x direction
j	Gridblock position in y direction
k	Iteration number
o	Oil
w	Water

Superscripts

n	Time level ($n =$ old time level, $n + 1 =$ new time level)
-----	--

REFERENCES

1. Amdahl, G.: "Validity of the Single-Processor Approach to Achieving Large-Scale Computer Capabilities," *AFIPS Conf. Proc.* **30**, (1967) 483-85.
2. Appleyard, J.R. and Cheshire, I.M.: "Nested Factorization," paper SPE 12264, presented at the SPE Symposium on Reservoir Simulation, San Francisco (Nov. 15-18, 1983).
3. Aziz, K. and Settari, A.: *Petroleum Reservoir Simulation*, Applied Science, London (1979).
4. Bard, Y., *Nonlinear Parameter Estimation*, Academic Press, New York (1974).
5. Barua, J. and Horne, R.N.: "Computerized Analysis of Thermal Recovery Well Test Data," *SPE Formation Evaluation* (Dec. 1987) 560-67.
6. Barua, J., Kucuk, F. and Gomez-Angulo, J.: "Application of Computers in the Analysis of Well Tests from Fractured Reservoirs," paper SPE 13662, presented at the SPE California Regional Meeting, Bakersfield (Mar. 27-29, 1985).
7. Behie, A.: "Comparison of Nested Factorization, Constrained Pressure Residual, and Incomplete Factorization Preconditionings," paper SPE 13531, presented at the SPE Symposium on Reservoir Simulation, Dallas (Feb. 10-13, 1985).
8. Behie, A. and Forsyth, P.A., Jr.: "Incomplete Factorization Methods for Fully Implicit Simulation of Enhanced Oil Recovery," *SIAM J. Sci. Stat. Comp.*, **5** (1984) 543-61.
9. Behie, A. and Vinsome, P.K.W.: "Block Iterative Methods for Fully Implicit Reservoir Simulation," *SPEJ* (Oct. 1982) 659-68.
10. Bentsen, R.G. and Donohue, D.A.T.: "A Dynamic Programming Model of the Cyclic Steam Injection Process," *JPT* (Nov. 1969) 1582-96.
11. Bertiger, W.I. and Kelsey, F.J.: "Inexact Adaptive Newton Methods," Paper SPE 13501, presented at the SPE Symposium on Reservoir Simulation, Dallas (Feb. 10-13, 1985).
12. Boberg, T.C. and Lantz, R.B.: "Calculation of the Production Rate of a Thermally Stimulated Well," *JPT* (Dec. 1966) 1613-23.
13. Boyle, J., Butler, R., Disz, T., Glickfeld, B., Lusk, E., Overbeek, R., Patterson, J. and Stevens, R.: *Portable Programs for Parallel Processors*, Holt, Rinehart, Wingston, Inc. New York (1987).
14. Brown, P.N., Hindmarsh, A.C. and Walker, H.F.: "Experiments with Quasi-Newton Methods in Solving Stiff ODE Systems," *SIAM J. Sci. Stat. Comp.*, **6** (1985) 297-313.
15. Broyden, C. G.: "A Class of Methods for Solving Nonlinear Simultaneous Equations," *Math. Comp.*, **319** (1965) 577-93.

16. Broyden, C. G.: "The Convergence of a Class of Double Rank Minimization Algorithms. 2. The New Algorithm," *J. Inst. Math. Appl.*, **6** (1970) 221-31.
17. Concus, P., Golub, G.H. and O'leary, D.P.: "A Generalized Conjugate Gradient Method for the Numerical Solution of Elliptic Partial Differential Equations," Computer Science Dept., Stanford University, STAN-CS 76-533 (Jan. 1976).
18. Dembo, R.S., Eisenstat, S.C. and Steihaug, T.: "Inexact Newton Methods," *SIAM J. Num. Anal.*, **18** (1982) 400-408.
19. Dennis, J. E. and Schnabel, R. B.: *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Englewood Cliffs, New Jersey (1983).
20. Dubois, P.F., Greenbaum, A. and Rodrigue, G.H.: "Approximating the Inverse of a Matrix for Use in Iterative Algorithms on Vector Processors," *Computing*, **22** (1979) 257-68.
21. Earlougher, R.C., Jr.: *Advances in Well Test Analysis*, SPE Monograph, Vol. 5, Chapter 12 (1977).
22. Efrat, I. and Tismenetsky, M.: "Parallel Iterative Linear Solvers for Oil Reservoir Models," *IBM J. Res. Develop.* (Mar. 1986) 184-92.
23. Forsyth, P., Jr. and Sammon, P.H.: "Gas Phase Appearance and Disappearance in Fully Implicit Black Oil Simulation," *SPEJ* (Oct. 1984) 505-507.
24. Gill, P.E., Murray, W. and Wright, M.H.: *Practical Optimization*, Academic Press, New York (1983).
25. Gill, P.E. and Murray, W.: "Quasi-Newton Methods for Unconstrained Optimization," *J.I.M.A.*, **9**, (1972) 91-108.
26. Gontijo, J.E. and Aziz, K.: "A Simple Analytical Model for Simulating Heavy Oil Recovery by Cyclic Steam in Pressure-Depleted Reservoirs," paper SPE 13037, presented at the SPE Annual Technical Conference and Exhibition, Houston (Sep. 16-19, 1984).
27. Greenstadt, J.L.: "On the Relative Efficiencies of Gradient Methods," *Math. Comp.*, **21** (1967) 360-367.
28. Gringarten, A.C., Bourdet, D.P., Landel, P.A. and Kniazeff, V.J.: "A Comparison Between Different Skin and Wellbore Storage Type-Curves for Early-Time Transient Analysis," paper SPE 8205, presented at the SPE Annual Technical Conference and Exhibition, Las Vegas (Sep. 23-26, 1979).
29. Guillot, A.Y. and Horne, R.N.: "Using Simultaneous Flow Rate and Pressure Measurements to Improve Analysis of Well Tests," *SPE Formation Evaluation* (June 1986) 217-226.
30. Gustafson, J. L., Montry, G. R. and Benner, R. E.: "Development of Parallel Methods for a 1024-Processor Hypercube," *SIAM J. Sci. Stat. Comp.*, **9**, (1988) 609-38.

31. IMSL Inc., *IMSL MATH/LIBRARY User's Manual*, IMSL, Houston (1987).
32. Kelley, C.T. and Sachs, E.W.: "A Quasi-Newton Method for Elliptic Boundary Value Problems," *SIAM J. Num. Anal.*, **24**, (1987) 516-31.
33. Killough, J. E. and Wheeler, M. F.: "Parallel Iterative Linear Equation Solvers: An Investigation of Domain Decomposition Algorithms for Reservoir Simulation," paper SPE 16021, presented at the SPE Symposium on Reservoir Simulation, San Antonio (Feb. 1-4, 1987).
34. Lasdon, L., Coffman, P.E., Jr., MacDonald, R., McFarland, J.W. and Sepehrnoori, K.: "Optimal Hydrocarbon Reservoir Production Policies," *Operations Research*, **34** (1986) 40-54.
35. Levenberg, K.: "A Method for the Solution of Certain Problems in Least Squares," *Quart. Appl. Math.*, **2**, (1944) 164-68.
36. Marquardt, D.W.: "An Algorithm for Least-Squares Estimation of Nonlinear Parameters," *J. Soc. Indust. App. Math.*, **11**, (1963) 431-41.
37. Mattheis, H. and Strang, G.: "The Solution of Nonlinear Finite Element Equations," *Int. J. Num. Meth. Engg.*, **14**, (1979) 1613-26.
38. McEdwards, D.G.: "Multiwell Variable-Rate Well Test Analysis," *SPEJ* (Aug. 1981) 444-46; *Trans., AIME*, **271**.
39. Nanba, T. and Horne, R.N.: "An Improved Regression Algorithm for Automated Well Test Analysis," paper SPE 18161, presented at the SPE Annual Technical Conference and Exhibition, Houston (Oct. 2-5, 1988).
40. Nghiem, L.X.: "A New Approach to Quasi-Newton Methods With Application to Compositional Modelling," paper SPE 12242, presented at the SPE Symposium on Reservoir Simulation, San Francisco (Nov. 16-18, 1983).
41. Odeh, A.: "Comparison of Solutions to a Three-Dimensional Black-Oil Reservoir Simulation Problem," *JPT* (Jan. 1981) 13-25.
42. Padmanabhan, L. and P.T. Woo: "A New Approach to Parameter Estimation in Well Testing," paper SPE 5741, presented at the 1976 SPE Symposium on Reservoir Simulation, Los Angeles (Feb. 19-20).
43. Padmanabhan, L.: "Welltest - A Program for Computer-Aided Analysis of Pressure Transients Data from Well Tests," paper SPE 8391, presented at the SPE Annual Technical Conference and Exhibition, Las Vegas (Sep. 23-26, 1979).
44. Padua, D.A. and Wolfe, M.J.: "Advanced Compiler Optimizations for Supercomputers," *Comm. ACM*, **29** (1986) 1184-1200.
45. Prats, M.: *Thermal Recovery*, SPE Monograph Vol. 7, Chapter 9 (1977).

46. Rivero, R.T. and Heintz, R.C.: "Resteaming Time Determination - Case History of a Steam Soak Well in Midway Sunset," *JPT* (June 1975) 665-671.
47. Rosa, A.J. and Horne, R.N.: "Automated Type-Curve Matching in Well Test Analysis Using Laplace Space Determination of Parameter Gradients," paper SPE 12131, presented at the SPE Annual Technical Conference and Exhibition, San Francisco (Oct. 5-8, 1983).
48. Scott, S. L., Wainwright, R. L., Raghavan, R. and Demuth, H.: "Application of Parallel (MIMD) Computers to Reservoir Simulation," paper SPE 16020, presented at the SPE Symposium on Reservoir Simulation, San Antonio (Feb. 1-4, 1987).
49. See, B.A. and Horne, R.N.: "Optimal Reservoir Production Scheduling by Using Reservoir Simulators," *SPEJ* (Oct. 1983) 717-26.
50. Settari, A. and Aziz, K.: "A Generalization of the Additive Correction Method for the Iterative Solution of Matrix Equation," *SIAM J. Num. Anal.*, **10** (1973) 506-21.
51. Stehfest, H.: "Algorithm 368, Numerical Inversion of Laplace Transforms," D-5, *Comm. ACM*, **13**, (1970) 47-48.
52. Steihaug, T.: "Quasi-Newton Methods for Large Scale Nonlinear Problems," Ph.D. Thesis, School of Organization and Management, Yale University, New Haven, CT. (1981).
53. Tewarson, R.P. and Zhang, Y.: "Sparse Quasi-Newton LDU Updates," *Int. J. Num. Meth. Eng.*, **24**, (1987) 1093-1100.
54. Thomas, G.W. and Thurnau, D.H.: "Reservoir Simulation Using an Adaptive Implicit Method," paper SPE 10120, presented at the SPE Annual Technical Conference and Exhibition, San Antonio (Oct, 5-7, 1981).
55. Tsang, C.F., McEdwards, D.G., Narasimhan, T.N. and Witherspoon, P.A.: "Variable Flow Well Test Analysis by a Computer Assisted Matching Procedure," paper SPE 6547, presented at the SPE Annual California Regional Meeting, Bakersfield (April 13-15, 1977).
56. Van Lookeren: "Calculation Methods for Linear and Radial Steam Flow in Oil Reservoirs," *SPEJ* (June 1983) 427-39.
57. Vinsome, P. K. W.: "Orthomin, an Iterative Method for Solving Sparse Banded Sets of Simultaneous Linear Equations," paper SPE 5729, presented at the SPE Symposium on Reservoir Simulation, Los Angeles (Feb. 19-20, 1976).
58. Wallis, J.R.: "Incomplete Gaussian Elimination as a Preconditioner for Generalized Conjugate Gradient Acceleration, paper SPE 12265, presented at the SPE Symposium on Reservoir Simulation, San Francisco (Nov. 16-18, 1983).
59. Wallis, J.R., Kendall, R.P. and Little, T.E.: "Constrained Residual Acceleration of Conjugate Residual Methods," paper SPE 13536, presented at the SPE Symposium on Reservoir Simulation, Dallas (Feb. 10-13, 1985).

60. Watts, J. W.: "A Method of Improving Line Successive Overrelaxation in Anisotropic Problems - A Theoretical Analysis," *SPEJ* (Mar. 1971) 105-18.

