PRECONDITIONED NEWTON METHODS FOR SIMULATION OF RESERVOIRS WITH SURFACE FACILITIES

A DISSERTATION

SUBMITTED TO THE DEPARTMENT OF PETROLEUM ENGINEERING

AND THE COMMITTEE ON GRADUATE STUDIES

OF STANFORD UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

By

Thomas James Byer

May 2000

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Dr. Khalid Aziz (Principal Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Dr. Michael Edwards

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Dr. Roland Horne

Approved for the University Committee on Graduate Studies:

Abstract

The primary focus of this research is to develop methods for increasing the computational efficiency of fully coupled implicit reservoir and surface facility problems, which can utilize both sequential and parallel processing environments. This research also focuses on the object-oriented design of the application code with the goals of reusability and extendibility.

Explicit coupling is the most common method for simulation of reservoir flow with surface facilities. The coupling is through well boundary conditions and involves a sequentially implicit solve of the facilities and reservoir per timestep. This approach allows for simple coupling at the expense of material balance errors due to changes in the pressure and saturation distribution over the timestep.

A standard implicit formulation of the entire system will avoid material balance errors, however the number of Newton iterations generally increases, adding significantly to the simulation cost. In either the implicit or explicit formulation the quality of the initial estimate of the facility solution strongly affects the convergence behavior of the coupled system.

New methods for accelerating convergence of Newton's method for the fully coupled system are developed. In the explicit method the most significant material balance errors coincide with the transition period defined by the beginning of two-phase flow in the wellbore. Two new techniques are introduced:

1. An adaptive explicit coupling method is introduced that is designed to combine the efficiency of explicit coupling with the accuracy of the standard implicit formulation. The central idea involves switching between fully implicit coupled reservoir and facilities and explicit coupling depending on the occurrence of phase transitions at the wells. The switching criteria developed is based on detecting phase transitions at the wells so that the fully implicit facilities formulation can be employed, thereby minimizing material balance errors. Outside the transition zone an explicit

formulation is employed in order to avoid the more expensive full Newton iteration, however, this approach cannot completely eliminate the material balance error.

The new adaptive explicit coupling method is limited to problems where phase transition for all the wells occurs approximately at the same time. If the transition period for each well is distinct from the other wells, then the efficiency of the adaptive method reduces to that of the standard implicit method.

2. A new preconditioning method is presented that is designed to accelerate convergence of the standard implicit facilities formulation. The method can be applied with implicit and adaptive implicit treatment of the reservoir flow field. Additionally, when the error levels outside the transition periods are acceptable, the preconditioning method can be used to accelerate the standard implicit iterations required by the adaptive explicit facilities formulation. Test results show that the preconditioned method significantly reduces the cost of standard fully implicit coupling. When the preconditioned adaptive explicit coupling formulation is employed the cost of the formulation is very similar to that of explicit coupling, while minimizing material balance errors.

An analysis of the computational cost for each component of the preconditioning method shows that facilities Jacobian calculations are the most expensive. Two parallel models are developed for distributing the Jacobian calculations across multiple processors. The first model demonstrates excellent efficiency for two processors, however due to the task assignment algorithm, limited efficiencies are observed for more than two processors. The second model is designed to overcome the limitations of the first model and results in good efficiencies for up to six processors.

The complexity of the application code developed for this research demonstrates the viability of object-oriented methods. Several examples are presented that demonstrate the reusability of the object model data structure and computational methods. Additionally, the object and computational model design provide significant insights into the design of objects and methods required for the new preconditioning technique and parallel formulations.

Acknowledgments

I would like to express my gratitude and sincere appreciation to Dr. Khalid Aziz and Michael Edwards for their support, encouragement and guidance through the course of this study. The appreciation is extended to Dr. Roland Horne for participating on the reading and examination committees.

I am thankful to the Petroleum Engineering Department and the SUPRI-B research group at Stanford for the financial support provided throughout this research. I am grateful to Chevron Petroleum Technology Company for granting me leave to pursue the Ph.D. program at Stanford.

I would like to express my respect and gratitude to my parents, Leo Rzepiela, brother and sisters. Many thanks to them all for their support, encouragement and understanding throughout the years.

For their unwavering patience and love, I dedicate this dissertation to four remarkable women, my wife Lupe, and daughters Elizabeth, Alice, and Emily.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Reservoir simulation is perhaps the primary tool used for field development planning. Full field models assist in determining long term operating strategies and pattern models are used to study sweep efficiencies of various enhanced oil recovery processes. Detailed well models allow for investigation of optimal well orientation and assessment of individual well performance. The surface facility model is used to determine the operational requirements for the collection and processing of produced or injected fluids. In many fields the well performance and reservoir development are strongly influenced by the surface facility. Coupled reservoir and facility simulation allows for investigation of the combined system. The current economic environment requires better selection of which properties to develop and more precise development plans for existing properties. This increased focus on asset management coupled with the availability of high performance computing resources has elevated the need for extensible modeling tools and has expanded the scope of simulation.

At the reservoir level fine scale geological descriptions are generated from geostatistical programs. To assess uncertainty, several equiprobable realizations of reservoir properties are generated and used in flow simulators. The fine scale descriptions produce simulation models that are more intensive computationally to solve. The fine scale descriptions result in discretized models where the number of cells is on the order 10^5 and higher. The standard first order fully implicit formulation of the reservoir flow equations is unconditionally stable, however this method can have significant computational and memory requirements. Parallel solution methods have been constructed which seek to obtain scalability and distribute the memory load across several processing units. Other strategies based on the structure of the flow equations have been designed to reduce computational effort through reduced implicitness. Most notable are the implicit pressure explicit saturation (IMPES) and adaptive implicit formulations.

Field development planning may be influenced more strongly by surface facilities than by reservoir characteristics. The available separation capacity can limit field production or injection. Separation units are designed to operate within a specified range of pressures and temperatures and proper assignment of wells to low or high-pressure separation units can maximize the total production. The separated water or gas may be used for injection programs that are constrained by available pump or compression horsepower. Integrated reservoir and surface facility models can capture these different scenarios to provide more reliable forecasts. Additionally, the facility model can also provide a tool for evaluating different gathering system options and for designing future development of injection or production schedules.

A common calculation approach for coupling the reservoir with surface facilities is to generate boundary conditions for the wells from an explicit treatment of the facility model. This approach allows for simple coupling at the expense of material balance errors due to changes in the pressure and saturation distribution over the timestep. Additionally, introduction of new wells into the production or injection system may require several iterations between the facility and reservoir model to resolve the new pressure and rate distributions adequately over a timestep.

A fully implicit formulation of the entire system will avoid material balance errors, however the number of Newton iterations per timestep can increase. In either formulation the quality of the initial estimate for the facility solution strongly affects the convergence behavior of the coupled system.

Associated with the requirements of comprehensive modeling tools is an increased complexity of simulation codes. Traditional structured programming languages such as Fortran do not provide enough constructs and data modeling flexibility to allow for an easily extensible code design. While the introduction of derived types in the Fortran 90 language has significantly enhanced data modeling capability, application of other object-orientated principles, such as inheritance, is limited. The strengths of these languages are in their ability to provide highly optimized code for developing efficient numerical algorithms.

Parallel computation potentially can reduce the computation time of simulation codes by a factor that is proportional to the number of processing nodes, however code development is more complex. The options available to incorporate parallel computing technology into simulation codes are driven primarily by whether the computing environment employs distributed or shared memory architecture.

The distributed memory architecture is defined by a set of computing nodes, with each node having local memory and instruction stream independent of the other nodes. Data and messages are exchanged via network connections or high performance switches. Since each node's local memory cannot directly access another node's local memory, parallelism is obtained by employing message passing libraries to distribute data and the computational tasks. In this context the developer is responsible for managing the parallel content of the algorithm. Many of the numerical algorithms that employ some form of domain decomposition have natural mappings to a distributed environment.

The shared memory architecture provides parallelism through multithreaded programming. In this context the same global address space is available to all the computing nodes through a common memory bus. A manager process with access to global memory space controls program execution and parallelism is achieved by spawning subprocesses that inherit part or all of the manager's address space. Since no explicit message passing functionality is required, application development is generally easier than in a distributed memory environment. The main benefit of this architecture is the high rate of data communication between processors. However, for problems with large memory requirements such as reservoir simulation, a common memory bus restricts individual node performance and hence total system scalability.

Both of these approaches can benefit from designing an application based on object-oriented techniques. For example, in a distributed memory environment, standardized objects for parallel vectors and matrices can be constructed to provide high level matrix vector operations and mask the low level functionality of message passing libraries. Similarly, in a shared memory environment standardized objects which manage processes, yet are independent of the specific task assigned to the process, can be designed to obtain a common mechanism for achieving and extending parallelism.

Chapter 2 provides a literature review of the relevant research in these areas and concludes with the focus of this dissertation. Chapters 3 and 4 present the physical and numerical models used in this research. Chapter 5 presents a new coupling and preconditioning method for solving reservoir and facility models. Chapter 6 extends this method to a parallel environment and Chapter 7 presents the object-oriented design used to develop the simulator used in this research. Conclusions and future areas of research are presented in Chapter 8.

Chapter 2

Literature Review

In this chapter the relevant works from the fields of reservoir and facility modeling are discussed. A more complete description of reservoir simulation topics can be found in Aziz and Settari (1979) or Peaceman (1977). Similarly for facility modeling, Jeppson (1977) provides a detailed analysis of network simulation and Govier and Aziz (1972) provide a detailed analysis of the flow of complex mixtures in pipes.

2.1 Reservoir Simulation - Adaptive Implicit Methods

Reservoir simulation involves the application of numerical methods to solve a system of conservation laws that describe flow in a porous medium. For a black oil isothermal formulation the independent variables are represented in terms of pressure and saturation. Several authors (Aziz and Settari, 1979, Peaceman, 1977, Bell *et al.*, 1986) have studied the structure of these equations. They have shown that: (1) the pressure equation is essentially parabolic or elliptic and must be treated implicitly and (2) the saturation equations are essentially hyperbolic and both explicit and implicit schemes are applicable.

Modeling displacement processes with a fully implicit formulation allows for large timesteps (unconditional stability) at the cost of inverting a fully implicit Jacobian. For large timesteps a standard first order fully implicit formulation can lead to significant smearing of the saturation profile and has a fully implicit Jacobian matrix to invert for each Newton iteration. IMPES (implicit pressure explicit saturation) schemes avoid the cost of full Jacobian matrix inversion (and even have a smaller truncation error coefficient compared to implicit schemes), however they are accompanied by a timestep restriction.

For example, in one dimension it is well known that stability of the single point upstream weighting scheme is governed by the condition

$$v\frac{\Delta x}{\Delta t} \leq 1 \tag{2.1}$$

where v is the fluid velocity, Δx is the distance along the x-direction, and Δt is the time interval. This inequality is equivalent to the well-known Courant-Friedrichs-Lewy (Richtmyer, 1957) condition for hyperbolic equations. For smaller timesteps with CFL below unity, an IMPES formulation is more efficient than a fully implicit formulation. However, the timestep size reduces with increasing velocity and decreasing mesh size, as can be seen from Eq. 2.1. Domains containing wells and high permeability streaks are classical examples where high velocities are obtained. Such cases can severely limit the timestep for an IMPES method, particularly for smaller mesh size.

Adaptive implicit methods have been developed that allow for an implicit formulation to be retained locally in high velocity regions, thereby removing the local timestep limitation, while the scheme is explicit (i.e. IMPES) elsewhere, so as to reduce total computational cost.

The fully implicit formulation involves calculation of the flux at the new time level $n+1$ while IMPES calculates pressure implicitly at time level $n+1$ and calculates transmissibility explicitly at time level *n.* The adaptive implicit formulation involves switching between the time levels of the respective fully implicit and IMPES formulations according to the local CFL condition. At the cell interface between implicit and explicit cells the phase transmissibility must be defined to ensure stability and maintain local conservation (this is further discussed in Section 3.3.2).

Several methods have been proposed to determine the level of implicitness in the formulation. The earliest approach by Thomas *et al.* (1983) was based on criteria derived empirically depending upon maximum changes in the primary variables. If the saturation or pressure change in an explicit grid cell exceeds a predefined tolerance the saturation variable is treated implicitly. Forsyth and Sammon (1986) presented cases where this tolerance is not sufficient for stability. Subsequent approaches have been based on the CFL condition for stability, see Fung *et al*. (1989), Russell *et al*. (1989), Grabenstetter *et al*. (1991). The approaches differ in the method for computing the characteristic velocity.

Chien and Northrup (1993) present an adaptive implicit method that allows for static implicit or explicit regions coupled with dynamic local grid refinement. The adaptive implicit formulation is obtained by allowing refined regions to change in time and by specifying the region as either implicit or IMPES.

2.2 Reservoir Simulation - Domain Decomposition Methods

The aim of a decomposition technique is to subdivide a problem into a set of smaller problems that are easier to solve. Decomposition techniques have been employed in a variety of contexts for solving linear and nonlinear systems of equations (Smith *et al.* 1996, Killough and Wheeler, 1997). When the governing equations or geometry define logical boundaries in the global problem, domain decomposition provides formalism for defining solution domains and treatment of domain boundary conditions. In the context of nonlinear problems, the domains are solved via Newton's method, providing updated boundary conditions to neighboring domains.

Domain decomposition methods have an important role in the solution of simulation models. The complexity of reservoir models requires efficient methods to capture important flow behavior and to solve large systems of linear equations. Local grid refinement (LGR) methods are designed to provide accurate resolution of high flow gradients such as near well coning problems and capturing steep saturation fronts. Decomposition techniques applied at the matrix or physical level are commonly used in LGR formulations and are outlined below.

Wasserman (1987) developed a local grid refinement formulation that used the refined region to define a matrix decomposition of the Jacobian for the global coarse and fine grid problem. The coarse cell components are ordered first, followed by the fine cell components, thus maintaining a banded structure in matrix domains defined by the coarse and fine grid regions. The Schur matrix method (Golub and Ortega, 1993) is employed to solve the linear system and is constructed by applying a block Gaussian elimination to the coupled fine and coarse grid system. For example, if A_f and A_c represent the fine to fine and coarse to coarse grid Jacobian coefficients, respectively, the coupled matrix is defined by

$$\begin{bmatrix} A_f & A_{fc} \\ A_{cf} & A_c \end{bmatrix} \begin{bmatrix} \vec{X}_f \\ \vec{X}_c \end{bmatrix} = \begin{bmatrix} \vec{R}_f \\ \vec{R}_c \end{bmatrix} \tag{2.2}$$

where A_{fc} and A_{cf} represents coupling terms between coarse and fine regions, and the Schur system is given by

$$\hat{A}\vec{X}_f = \vec{R}_f - A_{fc}A_c^{-1}R_c \tag{2.3}$$

where $\hat{A}$ is called the Schur complement or capacitance matrix and is defined by

$$\hat{A} = A_f - A_{fc}A_c^{-1}A_{cf} \tag{2.4}$$

A similar system is constructed for the coarse grid. The coupling of the coarse and fine grids is based on extending the fine grid lines into the coarse grid and generating pseudocells with interpolated pressures and assigned saturations. The fine grid system is solved first subject to Dirichlet boundary conditions imposed on the pseudo cells. The coarse grid solution is obtained by back-substituting the fine grid solutions and solving the Schur system. The fine grid solution is updated using the new coarse grid solution. Through the application of structured linear equation solvers, domain decomposition at the matrix level allows for efficient solution of the combined system when compared to solving the fully coupled model where the banded structure is destroyed.

Methods have been designed to accelerate convergence of the linear solve by considering the coupling between the coarse and fine grid regions, in addition to the matrix structure properties. The basis for these approaches is the BEPS method developed by Bramble *et al*. (1988). A composite grid is created by superimposing the refined region onto coarse pseudocells that are aligned with the global coarse grid. The composite grid defines a coarse grid problem that is used to accelerate convergence of the fine grid problem. Wallis *et al*. (1993) used an algebraic approach to compute the Jacobian matrix coefficients for the pseudocells and demonstrated that in highly heterogeneous models the flow directions should be considered when constructing the matrix. Chien *et al*. (1993) present a LGR method that uses a similar coarsening technique to that of Wallis. As in the approach by Wasserman (1987), Schur complement methods can be applied to provide high-level matrix decomposition of the composite system.

Nacul (1991) applied domain decomposition at the reservoir level as an alternate approach to solving LGR models. Dirichlet and/or Neumann boundary conditions are specified at the domain interfaces. The solution technique consists of global iteration, which requires separate solution of each nonlinear problem defined by the coarse and refined regions, and then updating domain boundary conditions and checking for convergence. If convergence is not obtained, the domain problems are resolved with the updated boundary conditions. For the models tested the fixed pressure condition at the boundaries yielded more favorable results than the fixed rate condition when compared with respect to CPU time. The application of a preconditioner in the form of an IMPES solve on the coarse grid improved coarse-to-fine grid boundary conditions and indicated significant performance improvements in terms of reduced number of global iterations.

The merit of the reservoir level decomposition employed cannot be completely evaluated due to the linear solution technique used in Newton's method. A direct solver was employed which requires $O(n^3)$ operations and therefore would severely influence the CPU timings. In this scenario domain decomposition of any form has the possibility to reduce the computational requirements. Additionally, no comparisons with the BEPS method were presented.

Deimbacher *et al*. (1995) introduced a windowing technique for dynamic grid selection and placement. The objective is to locally apply gridding schemes in regions where increased accuracy is required. The base grid is always defined in Cartesian coordinates and the windowing regions are placed on the base grid as required. Neumann boundary conditions are used at the window interfaces and are fixed throughout the timestep. A global timestep size is selected and local timesteps are applied within the regions. A well coning example demonstrated the increased accuracy and efficiency of the windowing technique compared to traditional gridding techniques with global timesteps.

Similar to the LGR methods, domain decomposition techniques can be applied at the linear solve or reservoir level and are designed to exploit parallelism. The trend towards incorporating more geologic features into the simulation model through geostatistical methods has resulted in problems whose memory requirements exceed that usually available on a single workstation. Currently most research is focused on matrix level approaches where scalable formulations have been reported, for example, see Wallis *et al*. (1991), or Chien *et al*. (1997). The windowing technique presented by Deimbacher *et al*. (1995) may be suited for parallelism where local timesteps in each window may be performed concurrently. However no results have been presented that demonstrate the performance of this type of approach.

2.3 Coupled Reservoir and Facility Modeling

The mathematical model for an integrated system requires that the boundary conditions be specified via the surface facility. Typically this is at the low- or high-pressure separator, or at a source/sink node. Methods for coupled models have been developed that combine the individual algorithms with varying degrees of implicitness. This section begins with a brief review of the traditional method for solving surface facility problems. The methods for reservoir and surface facility coupling are then presented.

2.3.1 Surface Facility Modeling

The evaluation of facility models has evolved from research performed on the fundamental problem of water distribution systems, (Shamir *et al*., 1968). The water distribution network consists of pipes, pumps, valves and a set of nodes representing the intersection of pipes. The flows and pressure drops in the network are defined by nonlinear equations arising from analytical or experimental relationships. The method used for solving the facility system depends on whether closed loops are allowed in the network structure, which are common features in gas or water gathering and distribution systems. This research is focused on surface facility configurations that arise in oilfield production and injection systems and therefore solution methods for closed loop systems will not be considered. The interested reader is referred to Wood *et al*. (1972), or Mucharam *et al*. (1990) for a discussion of closed loop systems.

Startzman *et al*. (1987) presented two methods for computing steady state pressures and rates in a single-phase surface facility problem. The facility problem is first defined in terms of node and node connecting elements. A node represents a source, sink or a junction, and a node connecting element represents a flow element, such as a pipe or valve. The node connecting elements contribute to pressure change between two solution nodes and are not specified as solution nodes. The first method presented is a sequential technique that begins at nodes with specified boundary conditions. Unknowns at adjacent nodes are solved successively using the most recent solution estimate for the neighboring nodes. A solution estimate is obtained for every node in the network and the process is repeated until convergence. The second approach applies volumetric balances in residual form at each node. The residual equations are expanded in terms of the functional dependencies of the neighboring nodes and Newton's method is applied to the system of equations. A test problem with 73 solution nodes showed that the sequential method solved the problem about 20 times faster than the Newton's iterative method. Not enough data were presented to determine the source of the large difference.

2.3.2 Reservoir Facility Coupling

Most of the current methods for coupled reservoir and surface facility problems have basic components that can be traced to the approach presented by Emanuel *et al*. (1981). The formulation is based on an explicit coupling of the facility model and implicit treatment of the reservoir model. This is known to introduce material balance errors. The facility model is coupled to the reservoir model through the well equation. The well boundary condition is based on facility constraints and beginning of timestep reservoir conditions. Chapter 5 presents a detailed discussion of this formulation.

Schiozer (1994) studied solution methods for coupled reservoir and facilities models in three parts, reservoir, surface facilities, and the coupling between the systems. Within the reservoir model local grid refinement is used to improve the accuracy of near wellbore behavior and domain decomposition applied at the reservoir level, is used to improve the efficiency of LGR. The LGR and domain decomposition implementation was developed by Nacul (1991) and issues of the efficiency of the domain decomposition formulation were not investigated.

The facility model employed by Schiozer allowed for multiphase flow in pipe and choke specification. The equations describing the network problem are reduced to one equation per well to simplify implementation. The network reduction method is presented in Appendix A and it is shown that the derivative terms in the linearized reduced equations are not valid for multiphase flow and may result in convergence problems.

In Schiozer's work the solution of the coupled system was investigated using four methods. The first two methods investigated were the traditional explicit and increased implicit treatment of the facilities model and are fully described in Chapter 5. The explicit approach resulted in considerable material balance errors when chokes are present in the system. In the second method the boundary conditions are updated at every Newton iteration and result in a fully implicit solution. The material balance errors are eliminated, but at the expense of increased computing time versus the explicit formulation.

The third method can be viewed in terms of domain decomposition. The reservoir and facility are separate domains, and domain boundaries are at the completed well blocks. To achieve a better quality solution in the facility domain, the third approach placed the boundary conditions further into the reservoir, creating well subdomains. The method results in a fully implicit solution as shown in Fig 2.1. The final method examined was a standard fully implicit formulation at the matrix level of the entire system.

Schiozer used three test problems to compare the performance of the domain decomposition and standard fully implicit method. The base comparison for each is the

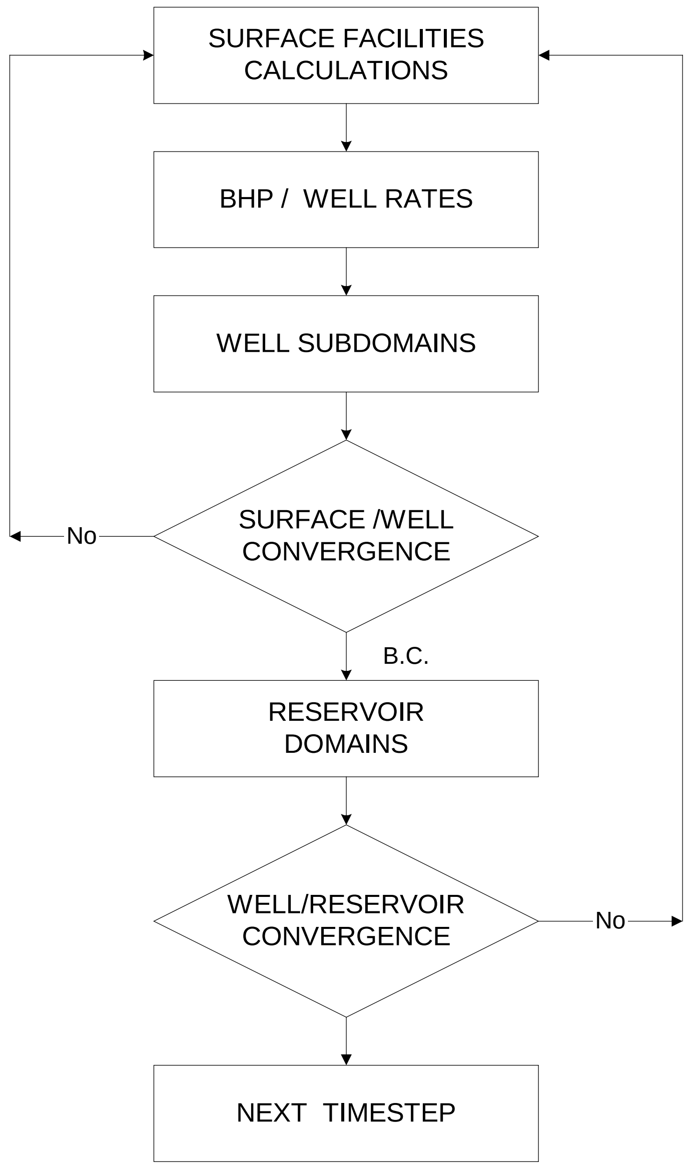

SURFACE FACILITIES CALCULATIONS
BHP / WELL RATES
WELL SUBDOMAINS
SURFACE /WELL CONVERGENCE
No
B.C.
RESERVOIR DOMAINS
WELL/RESERVOIR CONVERGENCE
No
NEXT TIMESTEP

Figure 2.1: Domain Decomposition Method developed by Schiozer (1991)

CPU time of explicit method, however a more appropriate baseline is the CPU time of the standard implicit formulation. The first problem contained only one well in the facility model. The results show that the standard implicit method requires approximately the same CPU time as the explicit method, but the domain decomposition method required 80% more time than the explicit method. The second and third problems contained four wells in the facility model, and third problem also included chokes. For these two problems the standard implicit method required 30-40% more CPU time than the explicit method, but 30-50% less time than the domain decomposition method. Thus, it was demonstrated that the most efficient method is simply a standard implicit formulation. It was suggested that as the complexity of the facilities model increased, the performance of the standard implicit method would degrade and the domain decomposition approach would become more efficient, however this was not demonstrated.

Litvak *et al*. (1997) presented results of an integrated study of the Prudhoe Bay surface pipeline network and reservoir. In their approach the integrated model consisted of several modules. The reservoir module is a compositional simulator. The well tubing string and surface pipeline network modules are used to evaluate well performance. The fluid separation module uses a compositional formulation and allows for complex separation configurations. Several other modules were implemented to determine assignment of voidage replacement and miscible injection schedules. The paper did not present a detailed description of the solution algorithm, but the rigorous treatment of the separation module and heuristic nature of injection schedules, suggests that the modules are solved independently with boundary condition updates from the connecting modules.

Hepguler *et al*. (1997) presented results that focus on implementation in a parallel processing environment. Their model used explicit couplings of commercial reservoir and network simulators where each simulator used a separate processor. The paper presented no new results related to numerical procedures and did not address load-balancing issues that are critical for the evaluation of any parallel implementation.

2.4 Object-Oriented Simulator Design

The primary objective of object-oriented modeling is to describe and specify a system that leads to a high quality software program. A quality software program has major features of correctness, reliability, and extendibility as described by Meyer (1988).

The application of object-oriented design (OOD) methods in reservoir and surface facility modeling has been explored only recently by industry and academia. The high cost of developing an application framework using an OOD paradigm has positioned the technology as proprietary within industry and the only published designs have been available through universities.

Nogaret (1996) developed a three-phase black oil simulator, SPARTA, based on a connection based approach presented by Lim *et al*. (1995) where the problem domain is defined in terms of a network of nodes and connections. This approach focuses on generation of the Jacobian and residual terms through looping over connections based on the flux definition. The base connection class defined virtual methods whose actual implementation is defined by derived classes and whose instantiation is through dynamic runtime binding. The derived classes defined the type of connection statically in terms of the phases present on each end of the connection and by the type of connection, reservoir to reservoir, or reservoir to facility. An analogous class hierarchy is defined for the transmissibility, well model, and pipe flow objects. It was shown that by abstracting the low-level functionality required for generating and assembling Jacobian terms, new types of connections can be easily implemented.

Verma (1996) developed a two-phase reservoir simulator, FLEX, for research on flexible gridding schemes. Connection based concepts were extended to accommodate multipoint flux schemes. The class hierarchy is based strongly on intercomponent relationships, “is a”, and intracomponent, “has a”, as defined by Booch (1994). The “is a” relationship implies inheritance while the “has a” relationship implies encapsulation. For example, since a sparse matrix is a type of matrix, the sparse matrix class is derived from the base matrix class, and since a reservoir has reserves, the *Reservoir* object encapsulates a *Reserves* object. The entire simulation problem is decomposed into the most elementary level based on these intracomponent and intercomponent relationships. By decomposing the complex system into smaller systems, common mechanisms are reused which allows for new functionality to be easily implemented.   This feature was quite evident due to the variety of gridding schemes investigated by Verma.

Parashar *et al*. (1997) developed a compositional reservoir simulator based on an object-oriented framework. The problem-solving environment is divided into three main functions: 1) application development interface for model and algorithm definition, 2) visualization, 3) input and output.   Each of these systems has access to lower level components and modules such as linear solvers and interpolation routines. The components and modules utilize low-level methods through an application specific semantics. The lowest level is the Hierarchical Distributed Dynamic Array which provides basic data management infrastructure for sequential and parallel data. These low-level tasks are encapsulated in programming abstractions for use by the components and modules. The decomposition and abstraction of the simulation components reduced the program complexity and enabled development of a flexible and parallel reservoir simulator.

2.5 Concluding Remarks

The literature review leads to the following observations:

- A variety of techniques can be used to solve coupled reservoir and surface facility models. However, no method has been shown to be universally applicable. Thus far the fully implicit formulation appears to be more efficient than the current domain decomposition formulations.
- The network reduction method is not applicable to multiphase flow in pipe and can result in convergence problems.
- Only one method has been presented for solving coupled reservoir and surface facility problems in parallel mode. The method relies on an explicit coupling of

the reservoir and facility models, and issues concerning scalability are not addressed.

- The object-oriented design used by Nogarat (1996) achieved a high level of abstraction by defining the problem domain in terms of nodes and connections. However, the static encoding of functionality in connection classes resulted in several classes that were nearly identical.
- The object-oriented design used by Verma (1996) was influenced strongly by the class relationships which lead to a logically organized yet overly complicated design. Numerous levels of nested component objects were used to construct and define composite objects and resulted in a composite object design with a high degree of internal object coupling that is difficult to understand without an in-depth knowledge of the entire class structure.

In this dissertation, methods designed to increase the computational efficiency of implicitly coupled reservoir and surface facility problems are described. The facility and reservoir models used in the coupled formulation are presented first. Then new coupling and preconditioning methods are presented. The new coupling method reduces the material balance errors associated with the explicit coupling. The preconditioning method is shown to accelerate convergence of fully implicit iterations and is developed for sequential and parallel modes. Additionally, this research presents object-oriented techniques that contain a strong computer science component and which provide an alternate characterization of the role of object-oriented technology applied to simulation problems when compared to the approaches described previously.

Chapter 3

Reservoir Model

This chapter presents the mathematical model, numerical formulation and solution methods used in the reservoir simulator developed for this research. The mathematical model describes the nonlinear partial differential equations governing multiphase flow in porous media. The numerical formulation describes the spatial and temporal operators employed for discretization. Newton's method forms an integral component of the solution strategy when employing fully implicit and adaptive implicit formulations of the highly nonlinear conservation equations.

3.1 Mathematical Model

Four basic principles are used to derive the mathematical model of flow in porous media: (1) conservation of mass, (2) conservation of energy, (3) fluid velocity relationships, and (4) an equation of state that relates pressure to density and temperature. This section provides a review of these principles and simplifications applied in the model. The conservation of energy in not discussed since only isothermal problems are considered.

3.1.1 Mass Conservation Equations

Mass conservation equations are required to model the simultaneous flow of distinct phases that may have variable composition in the porous medium. The conservation equations for $c = 1, \ldots, n_c$ components and stationary control volume *V*, are derived from the mass continuity equation defined by

$$\int_V \frac{\partial}{\partial t}\sum_{p=1}^{n_p} x_{cp}\rho_p S_p \phi dV = -\oint_S \sum_{p=1}^{n_p} x_{cp}\rho_p \vec{v}_p \cdot \vec{n}\, dS - \int_V \sum_{p=1}^{n_p} \tilde{M}_{cp}\, dV \qquad (3.1)$$

where n_p is the number of phases, x_{cp} is the mass fraction of component *c* in phase *p*, ρ_p is the phase density, S_p is the phase saturation, and ϕ is the porosity. The left-hand side term represents the time rate of change of mass in the volume. The second term represents the net mass flow rate out of the control volume with the velocity vector $\vec{v}_p$ normal to the surface area *S*. Mass transfer due to diffusion and dispersion is omitted. The last term represents the change in mass within the volume due to injection where $\tilde{M}_{cp}$ is defined as the mass injection rate per unit volume. A similar term but with opposite sign is used to represent change in mass due to production.

Application of the Gauss divergence theorem allows Eq. 3.1 to be expressed as

$$\int_V \frac{\partial}{\partial t}\sum_{p=1}^{n_p} x_{cp}\rho_p S_p \phi dV = -\int_V \sum_{p=1}^{n_p} x_{cp}\rho_p \vec{v}_p dV - \int_V \sum_{p=1}^{n_p} \tilde{M}_{cp} dV \qquad (3.2)$$

The control volume is assumed to be arbitrary and the integrands can be set to zero to obtain the differential form

$$\frac{\partial}{\partial t}\sum_{p=1}^{n_p} x_{cp}\rho_p S_p \phi = -\sum_{p=1}^{n_p} \nabla \cdot \left(x_{cp}\rho_p \vec{v}_p \right) - \sum_{p=1}^{n_p} \tilde{M}_{cp} \qquad (3.3)$$

Evaluation of the full flow field requires application of Eq. 3.3 for each component in the system. This results in n_c component equations for each control volume. The following section presents the fluid model and basic assumptions regarding phase behavior.

3.1.2 Fluid Model

The characterization of reservoir fluids has a direct impact on the mathematical formulation of the reservoir model. Compositional fluid characterization requires component and energy balance equations coupled with fugacity relationships to determine phase equilibrium conditions. In this work a two component hydrocarbon system with an aqueous phase is studied and is commonly known as the black-oil model. The phase equilibrium relationships are simplified by assuming no mass transfer between the aqueous and hydrocarbon phases. The relationships between the compositional and black-oil fluid properties are developed in Aziz *et al.* (1995). The mass fraction and phase density at equilibrium conditions are given as

$$x_{cp} = \frac{\rho_c^*}{\rho_p} \frac{R_{cp}}{B_p} \tag{3.4}$$

$$\rho_p = \frac{1}{B_p} \sum_{c=1}^{n_c} \rho_c^* R_{cp} \tag{3.5}$$

where ρ_c^* is the density of component c at standard conditions, B_p is the formation volume factor, and R_{cp} is the solubility of component c in phase p. Substituting these relationships into Eq. 3.3 leads to

$$\sum_{p=1}^{n_p} \frac{\partial}{\partial t}\left(\frac{R_{cp}}{B_p} S_p \phi \right) = -\sum_{p=1}^{n_p} \nabla \cdot \left(\frac{R_{cp}}{B_p} \vec{v}_p \right) - \frac{1}{\rho_c^*} \sum_{p=1}^{n_p} \tilde{M}_{cp} \tag{3.6}$$

Note that Eq. 3.6 is a mass balance at standard conditions and is applied to the oil, gas, and water components. The black-oil formulation implemented for this research does not allow water solubility and the oil component in the oil phase is not allowed to vaporize into the gas phase, therefore solubility restrictions reduce to

$$R_{ow} = R_{gw} = R_{og} = 0 \tag{3.7}$$

In this formulation the gas conservation equation contains a solution-gas component contained in the oil phase and a free gas component contained in the gas phase. The rate of mass transfer between the phases is defined by the gas solubility in oil, R_{go}.

3.1.3 Darcy's Law

The conservation equation requires evaluation of the fluid velocity for each phase. Darcy's law provides an empirical relationship between the potential gradient and fluid velocity. For multiphase flow the generalized Darcy's law is used to account for the relative permeability each phase and is expressed by

$$\vec{v}_p = -\frac{kk_{rp}}{\mu_p}\left(\nabla p_p + \rho_p g \nabla D\right) \tag{3.8}$$

where k is assumed to be diagonal permeability tensor, full permeability tensors are not considered in this work, k_{rp} is the relative permeability of phase p computed from the normalized form (Aziz and Settari, 1979) of the Stone II model (Stone, 1973), μ_p is the phase viscosity, and D is the vertical depth. Combining Eqs. 3.3 and 3.8 yields

$$\sum_{p=1}^{n_p}\frac{\partial}{\partial t}\left(x_{cp}\rho_p S_p \phi\right) = \sum_{p=1}^{n_p}\nabla\cdot\left(x_{cp}\rho_p \frac{kk_{rp}}{\mu_p}\left(\nabla p_p + \rho_p g\nabla D\right)\right) - \sum_{p=1}^{n_p}\tilde{M}_{cp} \tag{3.9}$$

3.1.4 Auxiliary Relationships

This section presents the constraints required to complete the flow description. The constraint equations specify equilibrium conditions for phase compositions, volumes, and pressures within the control volume. For composition this is expressed by

$$\sum_{c=1}^{n_c} x_{cp} = 1 \tag{3.10}$$

The volume constraint in the black-oil formulation is expressed in terms of saturation by

$$\sum_{p=1}^{n_p} S_p = 1 \tag{3.11}$$

The pressure relationships in a three-phase system are

$$P_{cow} = p_o - p_w \tag{3.12}$$

and

$$P_{cgo} = p_g - p_o \tag{3.13}$$

where P_{cow} and P_{cgo} are the oil-water and gas-oil capillary pressures which are assumed to be functions of water and gas saturations, respectively.

In this work the initial equilibrium conditions within the reservoir are assumed defined in the model input. A complete description of initialization procedures used for reservoir simulation is presented in Palagi (1992).

3.2 Numerical Formulation

This section presents the numerical formulation of the nonlinear system of partial differential equations presented previously. The system cannot be solved exactly and approximate techniques must be applied. In this research the locally conservative finite volume method was applied on Cartesian grids. The type of temporal approximation for saturation derivatives leads to implicit, explicit finite difference equations (IMPES) or an adaptive implicit formulation (Forsyth and Sammon, 1986). The discretized equations are linearized using a truncated Taylor series expansion and the resulting system of equations is solved using Newton's method.

3.2.1 Mass Conservation Equations

The discrete approximation of the mass conservation equations is presented in this section. The conservation equations contain second order partial derivatives of pressure and first order partial derivatives in time that are approximated by finite differences. The pressure, fluid and rock properties in the grid block are defined at the cell center while flux terms are assigned properties at block boundaries and therefore require approximations based on the properties of the surrounding blocks. The treatment of boundary conditions is described in Section 3.2.4.

Flow Terms

The flow component in Eq. 3.9 represents the mass flux rate per unit volume. Considering flow only in the *x*-direction for a cell located at grid indices *ijk* and assuming no sources or sinks, Eq. 3.9 reduces to

$$V_{ijk} \cdot \sum_{p=1}^{n_p} \frac{\partial}{\partial t}\left(x_{cp}\rho_p S_p \phi\right) = \sum_{p=1}^{n_p} \frac{\partial}{\partial x}\left(\zeta_x \left(\frac{\partial p_p}{\partial x} + \rho_p g \frac{\partial D}{\partial x}\right)\right)_{ijk} \Delta x_{ijk} \tag{3.14}$$

$$\zeta_x = x_{cp}\rho_p \frac{\mathrm{k}_x A_x k_{rp}}{\mu_p} \tag{3.15}$$

where V_{ijk} is the cell bulk volume and $A_x = \Delta y \Delta z$. To simplify further notation the phase potential is defined as

$$\Phi_p = p_p + \rho_p g D \tag{3.16}$$

with spatial derivative defined as

$$\frac{\partial \Phi_p}{\partial x} \equiv \frac{\partial p_p}{\partial x} + \rho_p g \frac{\partial D}{\partial x} \tag{3.17}$$

For an arbitrary phase *p*, the right-hand side of Eq. 3.14 can be expressed as

$$\left(\frac{\partial f}{\partial x}\right)_{ijk} \Delta x_{ijk} = \frac{\partial}{\partial x}\left(\zeta_x \frac{\partial \Phi_p}{\partial x}\right)_{ijk} \Delta x_{ijk} \tag{3.18}$$

where f represents the Darcy velocity and

$$f = \zeta_x \frac{\partial \Phi_p}{\partial x} \tag{3.19}$$

The divergence of the Darcy velocity is approximated with the discrete divergence operator, which for the one-dimensional Cartesian system shown in Fig. 3.1, is defined by the central difference operator

$$\left(\frac{\Delta f}{\Delta x}\right)_{ijk} = \frac{f_{i+1/2,j,k} - f_{i-1/2,j,k}}{\Delta x_{ijk}} \tag{3.20}$$

where f is evaluated at the cell boundaries:

$$f_{i\pm1/2,j,k} = \zeta_{x,i\pm1/2,j,k}\left(\frac{\Delta\Phi_p}{\Delta x}\right)_{i\pm1/2,j,k} \tag{3.21}$$

The pressure gradient components are approximated by the central difference operator to obtain

$$\left(\frac{\Delta\Phi}{\Delta x}\right)_{i-1/2,j,k} = \frac{\Phi_{i,j,k} - \Phi_{i-1,j,k}}{\Delta x_{i-1/2,j,k}} \tag{3.22}$$

where Φ is evaluated at the cell centers. A similar expression is defined for the $i+1/2, j,k$ cell face.

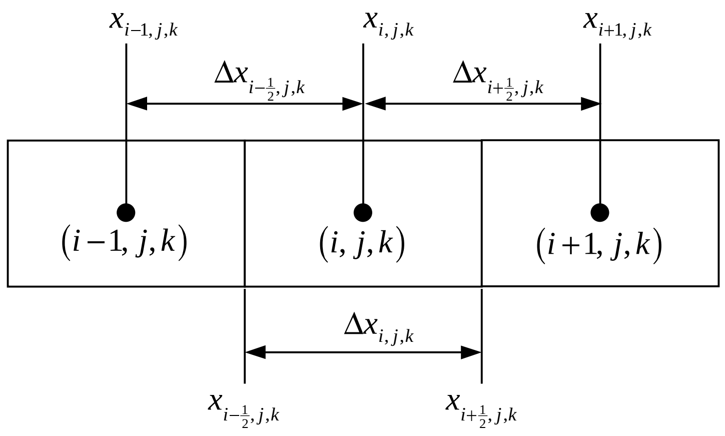


Figure 3.1: Block Centered Grid in One Dimension

Assuming uniform cell dimensions, application of these operators yields a second order accurate finite difference approximation for both the divergence of the Darcy velocity and pressure gradient components. The truncation error of the central difference approximation resulting from Eqs. 3.19-3.22 reduces from $O(\Delta x^2)$ to $O(\Delta x)$ for nonuniform grid spacing. Forsyth and Sammon (1988) demonstrated that for nonuniform grid spacing the cell centered approximation is convergent.

Utilizing Eqs. 3.18 through 3.22 allows the right-hand side of Eq. 3.18 to be approximated by

$$\Delta_x \left(\xi_x \Delta_x \Phi_p\right)_{ijk} \equiv \left(\xi_x\right)_{i+1/2,j,k}\left(\Phi_{i+1,j,k} - \Phi_{i,j,k}\right)_p - \left(\xi_x\right)_{i-1/2,j,k}\left(\Phi_{i,j,k} - \Phi_{i-1,j,k}\right)_p \tag{3.23}$$

where

$$\xi_x = \frac{\zeta_x}{\Delta x} \tag{3.24}$$

so that the discrete form of Eq. 3.14 can be defined by

$$\sum_{p=1}^{n_p} \Delta_x \left[\frac{\mathrm{k_x} A_x}{\Delta x} \frac{x_{cp}\rho_p k_{rp}}{\mu_p}\left(\Delta_x \Phi_p\right)\right]_{ijk} = \sum_{p=1}^{n_p} \Delta_x \left[\xi_x \left(\Delta_x \Phi_p\right)\right]_{ijk} \tag{3.25}$$

Phase transmissibility T_{cp}, can be defined as the product of a geometric factor T_g, and phase component $\tilde{T}_{cp}$ where

$$T_{cp,i+1/2,j,k} = \left(T_g \tilde{T}_{cp}\right)_{i+1/2,j,k} \tag{3.26}$$

$$T_{g,i+\frac{1}{2},j,k} = \frac{\left(\Delta y \Delta z\right)_{ijk}}{\dfrac{x_{i+1,j,k} - x_{i+1/2,j,k}}{k_{x,i+1,j,k}} + \dfrac{x_{i+1/2,j,k} - x_{i,j,k}}{k_{x,i,j,k}}} \tag{3.27}$$

$$\tilde{T}_{cp,i\pm 1/2,j,k} = \left(\frac{x_{cp}\rho_p k_{rp}}{\mu_p}\right)_{i\pm 1/2,j,k} \tag{3.28}$$

The geometric factor is a result of imposing flux continuity across the cell face. Pedrosa (1983) presented a detailed derivation for the geometric factors for both Cartesian and cylindrical systems. Generalization to full tensor permeabilities is presented in Edwards and Rogers (1994). The treatment of the transmissibility phase component is further discussed in Section 3.2.3.

The flow component for the one-dimensional Cartesian system shown in Figure 3.1 can be expressed by substituting Eq. 3.26 into 3.25 to obtain

$$\Delta_x \sum_{p=1}^{n_p} \left[\frac{\mathrm{k}_\mathrm{x} A_x}{\Delta x} \frac{x_{cp}\rho_p k_{rp}}{\mu_p} \left(\Delta_x \Phi_p \right) \right]_{ijk} = \sum_{p=1}^{n_p} \Delta_x \left(T_{cp} \Delta_x \Phi_p \right)_{ijk} \tag{3.29}$$

Generalizing the notation shown in Eqs. 3.23 and 3.24 to three dimensions yields

$$\Delta\left(\xi \Delta \Phi_p\right)_{ijk} \equiv \Delta_x \left(\xi_x \Delta_x \Phi_p\right)_{ijk} + \Delta_y \left(\xi_y \Delta_y \Phi_p\right)_{ijk} + \Delta_z \left(\xi_z \Delta_z \Phi_p\right)_{ijk} \tag{3.30}$$

so that the flow component can be compactly represented as

$$\sum_{p=1}^{n_p} \Delta\left(T_{cp} \Delta \Phi_p\right)_{ijk} \tag{3.31}$$

Source/Sink Terms

The placement of wells in the reservoir model introduces source/sink terms in the conservation equations. Since the wellbore is typically much smaller than the well block, the produced/injected mass is assumed to be averaged over the entire control volume and allows the source/sink term in Eq. 3.9 to be expressed as

$$\sum_{p=1}^{n_p} \tilde{M}_{cp} V_{ijk} = \sum_{p=1}^{n_p} M_{cp,ijk} \tag{3.32}$$

where $M_{cp,ijk}$ is the mass rate of component *c* in phase *p* for block *ijk*.

The presence of a source or sink term due to well placement introduces a singularity into the system. Due to the nonlinear nature of the partial differential equations analytical point source/sink equations cannot be applied. The resolution of the singularity involves a combination of analytical and numerical methods commonly known as the well model. The well model used in this research is presented in Section 3.2.5.

Accumulation Term

The accumulation term in Eq. 3.3 represents the change in mass of a component due to compressibility and saturation changes in the control volume over time. The time derivative is approximated using a first order finite difference operator defined by

$$\frac{1}{\Delta t}\Delta_t \Psi = \frac{1}{\Delta t}\left(\Psi^{n+1} - \Psi^{n}\right) \tag{3.33}$$

where Ψ represents any time dependant property and n is the time level. The accumulation term is then approximated by

$$\frac{V_{ijk}}{\Delta t}\sum_{p=1}^{n_p}\Delta_t\left(x_{cp}\rho_p S_p \phi\right)_{ijk} = \frac{V_{ijk}}{\Delta t}\sum_{p=1}^{n_p}\left(x_{cp}\rho_p S_p \phi\right)_{ijk}^{n+1} - \left(x_{cp}\rho_p S_p \phi\right)_{ijk}^{n} \tag{3.34}$$

3.2.2 Finite Difference System

The conservation equations for each grid block can be obtained by substituting Eqs. 3.31, 3.32 , and 3.34 into Eq. 3.3 to yield

$$\frac{V_{ijk}}{\Delta t}\sum_{p=1}^{n_p}\left[\left(x_{cp}\rho_p S_p \phi\right)^{n+1} - \left(x_{cp}\rho_p S_p \phi\right)^{n}\right]_{ijk} = \sum_{p=1}^{n_p}\Delta\left(T_{cp}\Delta\Phi_p\right)_{ijk} - \sum_{p=1}^{n_p} M_{cp,ijk} \tag{3.35}$$

Through application of the volume constraint and capillary pressure relationships described in Section 3.1.3 the oil, water, and gas component equations may be summarized as

$$\frac{V}{\Delta t}\Delta_t\left[x_{oo}\rho_o\left(1 - S_w - S_g\right)\right] = \Delta\left[T_{oo}\left(\Delta p_o + \rho_o g \Delta D\right)\right] - M_{oo} \tag{3.36}$$

$$\frac{V}{\Delta t}\Delta_t\left[x_{ww}\rho_w S_w\right] = \Delta\left[T_{ww}\left(\Delta p_o - \Delta P_{cow} + \rho_w g \Delta D\right)\right] - M_{ww} \tag{3.37}$$

$$\begin{gathered}\frac{V}{\Delta t}\Delta_t\left[x_{gg}\rho_g S_g \phi + x_{go}\rho_o\left(1 - S_w - S_g\right)\phi\right] = \\ \Delta\left[T_{gg}\left(\Delta P_{cgo} + \Delta p_o + \rho_g g \Delta D\right)\right] + \Delta\left[T_{go}\left(\Delta p_o + \rho_o g \Delta D\right)\right] - M_{gg} - M_{go}\end{gathered} \tag{3.38}$$

to yield three equations with three unknowns, p_o, s_w, and s_g. Since the transmissibility terms are functions of these primary variables the system of equations is nonlinear. A discussion of the linearization techniques and implicit/explicit time level of the formulation of the unknowns is presented in Section 3.3.

In this research it was assumed that oil and water saturations are present in every reservoir block and therefore p_o and s_w can always be specified as primary variables for the oil and water equations, respectively. However if the reservoir block pressure is above the bubble point pressure, no free gas is present and the block is considered undersaturated. Under these conditions the primary variable for the gas equation is defined to be the bubble point pressure, p_{bp}. Once the reservoir pressure drops to the bubble-point pressure, the primary variable is switched to s_g. A reservoir block may undergo a series of undersaturated and saturated states due to water or gas injection schedules and requires variable bubble-point treatment of the fluid properties.

In this research the variable bubble point formulation developed by Stright *et al.*, (1977) was employed. Their paper includes a detailed description of primary variable selection logic for reservoir blocks that cross the bubble-point and is shown to capture the variation in saturation pressure due to changes in solubility.

3.2.3 Nonlinear Terms

As shown in Section 3.2 the phase transmissibilities, phase densities and other terms that are contained in the interblock flow terms are themselves functions of the primary variables. The nonlinear terms can be classified by their dependence on pressure or saturation. In the black-oil formulation the terms which depend on the pressure of one phase only include composition, density, viscosity, and porosity. The terms that depend on phase saturation include relative permeability and capillary pressure. Evaluation of the pressure and saturation dependencies for each of these terms shows that the nonlinearity in the flow term is primarily due to relative permeability.

The weighting of the pressure dependant components of the interblock flow terms employs upstream or midpoint weighting schemes at the block boundaries. In this work upstream weighing is employed and for the one-dimensional example shown in Fig. 3.1, may be expressed as (omitting the relative permeability)

$$\tilde{T}_{cp,i+\frac{1}{2}} = \begin{cases} \left(\dfrac{x_{cp}\rho_p}{\mu_p} \right)_{i+1} & if \;\; \Delta_x \Phi_p > 0 \\ \left(\dfrac{x_{cp}\rho_p}{\mu_p} \right)_{i} & if \;\; \Delta_x \Phi_p \le 0 \end{cases} \tag{3.39}$$

which can be shown to be a spatially first order accurate approximation via a Taylor series expansion.

The weighting of the saturation dependent terms in the interblock flow terms is very critical. Aziz and Settari (1979) demonstrated that while midpoint weighting of the relative permeability is a second order approximation, a physically incorrect solution may be obtained. In this work single point upstream is employed and may be expressed as

$$k_{rp,i+\frac{1}{2}} = \begin{cases} k_{rp,i+1} & if \;\; \Delta_x \Phi_p > 0 \\ k_{rp,i} & if \;\; \Delta_x \Phi_p \le 0 \end{cases} \tag{3.40}$$

which is also a spatially first order accurate approximation.

3.2.4 Boundary Conditions

The boundary conditions for the model incorporate external and internal conditions. The external boundary conditions define limits of the reservoir and may include modifications for aquifer support. The internal boundary conditions result from shale barriers, sources or sinks from well placement, sealing faults, or domain decomposition methods. Both external and internal specifications are typically implemented by defining flow rate across a boundary or pressure at a boundary, which corresponds to Neumann or Dirichlet type conditions, respectively. Neumann boundary conditions corresponding to no flow reservoir boundaries expressed by

$$\vec{v}_p \cdot \vec{n} = 0 \tag{3.41}$$

for each phase *p*. Specifying the unknowns at some location in the model forms Dirichlet boundary are conditions. Constant pressure boundaries are expressed by

$$p_p(x, y, z) = p_{rb}(t) \tag{3.42}$$

where $p_{rb}(t)$ may be constant with time or represent an aquifer model.

3.2.5 Well Model

Usually production or injection of fluids in reservoir simulation is represented in terms of sources or sinks. The requirements of the well model are to relate specified boundary conditions to flow through the surrounding reservoir blocks. When the well contains multiple completions the coupling of the well to the reservoir becomes more complicated. The rates and phase behavior of each completion must be incorporated into the well model. The following sections present the well model used in this work.

Basic Well Model

The volume of a reservoir well block is typically much larger than the volume of a wellbore and the block pressure is not a good approximation for the well flowing pressure. The requirements of the well model are to relate block unknowns to the well pressure. Considering the well shown in Fig. 3.2 the well equation for Darcy flow can be expressed by

$$q_c = \sum_l WI_l \sum_{p=1}^{n_p} \left(\beta_p \tilde{T}_{cp}\right)_l \left(p_p - p_{wb}\right)_l \tag{3.43}$$

where β_p is a binary variable representing phase selection, WI_l and $p_{wb,l}$ are the respective well index and wellbore pressure for completed layer *l*. The methods for computing the well index and wellbore pressure are further discussed in this section.

Peaceman (1978, 1993) presented several well index models based on the premise that the well block pressure is related to the wellbore at some equivalent radius r_o. The first model was based steady state conditions for wells located in square blocks with no permeability anisotropy. In the model with anisotropic permeability the productivity index is defined by

$$WI_l = \frac{k\Delta z_l}{\left(\ln\left(\frac{r_o}{r_w}\right) + s\right)_l} \tag{3.44}$$

$$k = \left(k_x k_y\right)_l^{1/2} \tag{3.45}$$

$$r_{0,l} = 0.28\left[\frac{\left[\left(k_y/k_x\right)^{1/2}\left(\Delta x\right)^2 + \left(k_x/k_y\right)^{1/2}\left(\Delta y\right)^2\right]^{1/2}}{\left(k_y/k_x\right)^{1/4} + \left(k_x/k_y\right)^{1/4}}\right]_l \tag{3.46}$$

where r_w is the wellbore radius, r_o is the equivalent wellbore radius, and s is the skin factor.

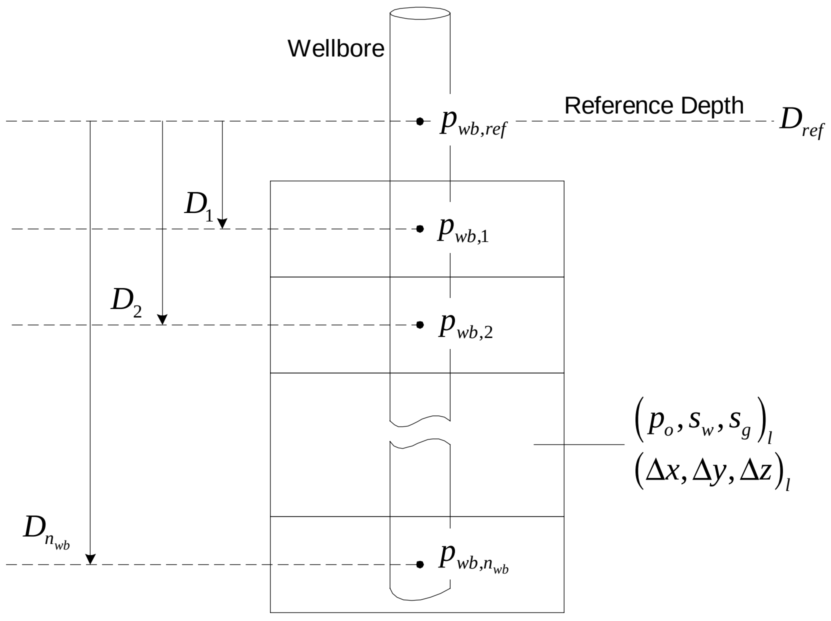


Figure 3.2: Schematic Representation of a Vertical Well

The wellbore pressure at the center of a completion is a function of the fluid column density and is typically expressed with respect to a reference pressure corresponding to the well bottom hole or pump location. Assuming that the reference depth is above the first completion, the wellbore density at a completion k is defined as

$$\rho_k = \frac{\sum_{l=1}^{k}\sum_{p=1}^{n_p}\left(\rho_p q_p\right)_l}{\sum_{l=1}^{k}\sum_{p=1}^{n_p}\left(q_p\right)_l} \tag{3.47}$$

where q_p is the phase flow rate. The wellbore gravity γ_k is then given by

$$\gamma_k = \gamma_{k-1} + 0.5g\left(\rho_k - \rho_{k-1}\right)\left(D_k - D_{k-1}\right), \quad k \neq 1 \tag{3.48}$$

where

$$\gamma_1 = \rho_1 g\left(D_1 - D_{ref}\right) \tag{3.49}$$

so that wellbore pressure can be expressed as

$$p_{wb,k} = p_{wb,ref} + \gamma_k \tag{3.50}$$

Implicit in this formulation is the assumption that tubing pressure losses due to friction and acceleration are negligible. Chapter 6 presents a method for coupling well and tubing models.

Well Constraints

Depending on the objectives of the study, well, group, or field level constraints may be imposed on the system. At the well level constraints are imposed via Neumann conditions for specifying flow rate or Dirichlet conditions for specifying wellbore pressure. Traditionally, group or field level constraints are implemented by resolving the constraints through application of individual well controls. Chapter 6 presents an alternative approach that allows well or group controls to be resolved via choke control.

Implementation of a well constraint requires specifying Eq. 3.43 as a source/sink term in the appropriate mass balance equation for each completed well block. Depending on the type of constraint an additional relationship may be required. When a mass flow rate, $q_{sp,c}$, is specified, the wellbore pressure is unknown in the well model and an additional equation is required to maintain a well-posed problem. The residual form of this equation is expressed by

$$r_{well,c} = \sum_{l} WI_l \sum_{p=1}^{n_p} \beta_p \tilde{T}_{cp,l} \left(p_p - p_{wb} \right)_l - q_{sp,c} \tag{3.51}$$

In the case of specified bottom hole pressure the flow rate is defined completely by existing variables and no additional relations are required. Also note that specification of this type of constraint implies some prior knowledge of the minimum flowing pressure and pressure losses in the tubing.

3.3 Solution of Nonlinear Problem

The system of discretized partial differential equations describing the flow field is highly nonlinear. This section describes the linearization and solution processes used in this research.

3.3.1 Fully Implicit Formulation

The fully implicit formulation is the most robust linearization method and the system (Eqs. 3.36-3.38) is solved commonly by Newton's method. This method involves an iterative sequence described in this section. The accumulation terms are expanded using conservative principles (Thomas, 1982) and the derivatives are evaluated at the current iteration level. The fully implicit treatment of these terms leads to a method that is unconditionally stable, allowing larger timesteps at the expense of larger truncation errors.

For cell *l* the conservation equation (Eq. 3.35) can be expressed in residual form as

$$R_{c,l}^{\upsilon+1} = \sum_{p=1}^{n_p} \left[\Delta \left(T_{cp} \Delta \Phi_p \right) \right]^{\upsilon+1} - \sum_{p=1}^{n_p} M_{cp} - \frac{V}{\Delta t} \sum_{p=1}^{n_p} \left[\left(x_{cp} \rho_p S_p \phi \right)^{\upsilon+1} - \left(x_{cp} \rho_p S_p \phi \right)^{n} \right] \tag{3.52}$$

where n is the old time level and υ is the current iteration level. Newton's method requires approximating the residual at iteration $(\upsilon+1)$ by their value at the current iteration plus a linear combination of the primary variables resulting from the partial differentiation of $R_{c,l}$ with respect to all the unknowns:

$$\begin{aligned} R_{c,l}^{\upsilon+1} \approx R_{c,l}^{\upsilon} + \sum_{m\in\Omega_l} \sum_{pv=1}^{n_{pv}} \left(\frac{\partial R_{c,l}}{\partial \overline{X}_{pv,m}} \right)^{\upsilon} \delta \overline{X}_{pv,m} + \\ \sum_{pv=1}^{n_{pv}} \left(\frac{\partial R_{c,l}}{\partial \overline{X}_{pv,l}} \right)^{\upsilon} \delta \overline{X}_{pv,l} + \sum_{pv=1}^{n_{pv}} \left(\frac{\partial M_{cp,l}}{\partial \overline{X}_{pv,l}} \right)^{\upsilon} \delta \overline{X}_{pv,l} \end{aligned} \tag{3.53}$$

where $\delta \overline{X}_{pv} = \overline{X}_{pv}^{\upsilon+1} - \overline{X}_{pv}^{\upsilon}$ and Ω_l is the set of all grid blocks connected to block l, and n_{pv} is the number of primary variables. In the black-oil formulation $n_{pv}=3$ and $\overline{X}_{pv} = \left[p_o, S_w, S_g / p_{bp} \right]$, where either S_g or p_{bp} is selected depending on presence of a gas phase. To estimate the unknowns at the current iteration level, $R_{c,l}^{\upsilon+1}$ is set to zero and Eq. 3.53 becomes

$$-R_{c,l}^{\upsilon} = \sum_{m\in\Omega_l} \sum_{pv=1}^{n_{pv}} \left(\frac{\partial R_{c,l}}{\partial \overline{X}_{pv,m}} \right)^{\upsilon} \delta \overline{X}_{pv,m} + \sum_{pv=1}^{n_{pv}} \left(\frac{\partial R_{c,l}}{\partial \overline{X}_{pv,l}} \right)^{\upsilon} \delta \overline{X}_{pv,l} + \sum_{pv=1}^{n_{pv}} \left(\frac{\partial M_{cp,l}}{\partial \overline{X}_{pv,l}} \right)^{\upsilon} \delta \overline{X}_{pv,l} \tag{3.54}$$

The linear system of equations can be compactly expressed as

$$A_r^{\upsilon} \delta \vec{X}_r^{\upsilon} = -\vec{R}_r^{\upsilon} \tag{3.55}$$

where A_r^{υ} is the Jacobian matrix. The system is solved for the changes in the primary variables and the iteration process is continued until $\left| R_{c,l}^{\upsilon} \right| < \varepsilon_R$, where ε_R is a specified tolerance.

3.3.2 Adaptive Implicit Formulation

Adaptive implicit methods employ a fully implicit formulation in high velocity regions, thereby removing the local timestep limitation, while an explicit IMPES formulation is employed elsewhere, so as to reduce total computational cost and increase accuracy away from high flow regions.

The fully implicit formulation involves calculation of the transmissibility at the new time level $n+1$ while IMPES involves calculation of the transmissibility at the old time level *n*. The adaptive implicit formulation involves switching between the time levels of the respective fully implicit and IMPES formulations according to the local CFL condition. For example, at the cell interface between implicit and explicit cells, where the left and right hand side cells have CFL numbers greater than unity and less than unity, respectively, the phase transmissibility is defined by

$$T_{p,i+\frac{1}{2}} = \begin{cases} T_{p,i}^{n+1} & if \;\; v_{p,i+\frac{1}{2}} > 0 \\ T_{p,i+1}^{n} & if \;\; v_{p,i+\frac{1}{2}} \le 0 \end{cases} \tag{3.56}$$

which ensures stability and maintains local conservation.

Several methods have been proposed to determine the level of implicitness in the formulation. In this work the CFL criteria developed by Russell (1989) was selected since it is based on quantities that are easily computed and does not require the calculation of the characteristic phase velocities. The CFL condition for stability of an explicit cell *l* may be expressed as

$$\Delta t \cdot \left(\frac{|v_x|}{\Delta x_i} + \frac{|v_y|}{\Delta y_i} + \frac{|v_z|}{\Delta z_i} \right) \le 1 \tag{3.57}$$

where in each direction the total velocity v, is defined by

$$v = \sum_{j=1}^{n_p} v_j \tag{3.58}$$

3.4 Concluding Remarks

The mathematical model and numerical formulation for the reservoir simulator developed for this research has been presented. Mass conservation equations for a two component hydrocarbon system with an aqueous phase result in a nonlinear system of partial differential equations. The locally conservative finite volume method is applied on

Cartesian grids. The treatment of the transmissibility allows for both implicit and adaptive implicit formulations. The numerical model was implemented in a simulator called FDS (Field Development System). Chapter 7 presents key design elements of the simulator.

Chapter 4

Surface Facility Model

An important component of field development simulation is the surface facility model that is used to identify the operational requirements and constraints based on factors such as individual well performance, gathering system configuration, and phase separation units. The surface facility model used in this research is presented in this chapter. The model is formulated in four components: 1) pipeflow, 2) choke, 3) separator, and 4) network. The pipeflow model predicts flow behavior and pressure profile in the wellbore and gathering or injection system. The choke model is used to maintain the well flow rate within safety limits of the equipment and to prevent gas and water coning. The separation model determines the oil and gas phases present at specified operating pressure and temperature. The network model is used to specify the overall facility connectivity that defines the flow path for produced or injected fluids.

The following sections present the formulation used in each of these components and where appropriate, examples of device performance or behavior is included. The chapter concludes with results from two test models that demonstrate the performance of the formulation and network solution strategy. The method for reservoir and surface facility coupling is presented in the following chapter and the effects the surface facility performance demonstrated in this chapter are further examined in the context of a coupled model.

4.1 Pipeflow Model

The formulation of the pipeflow model was based on the conservation of mass and momentum principles coupled with a phase behavior description. The results of experimental data have demonstrated that several flow regimes can exist in two-phase flow (Govier and Aziz, 1972) and that separate formulations for each regime are required. In this research a single pseudo-phase method with slip was employed to model steady state flow in pipe for an oil-gas system. The steady state formulation is justified since the objective of the coupled model is to capture field wide behavior in contrast to the transient models, which focus on pipeflow at much smaller time scale. For a detailed treatment of two-phase pipeflow models see Ouyang (1998).

The derivation of single-phase gas flow relationships is not included in this dissertation since the primary objective of the gas network is to provide pressure maintenance for extending the reservoir model production horizon. For a detailed treatment of single-phase gas flow see Ouyang and Aziz (1996).

4.1.1 Single Pseudo-Phase Model for Oil-Gas Flow

The single pseudo-phase approach assumes that the separate flowing phases can be modeled as a mixture without a distinct interface between phases. Average mixture properties are defined so that equations for single-phase flow can be utilized.

For a pipe with constant cross-sectional area A, the mixture density is defined by

$$\rho_m = \rho_o E_o + \rho_g E_g \tag{4.1}$$

where E_o and E_g are the oil and gas phase in-situ fractions, respectively, and ρ_o and ρ_g are the phase densities. The phase fraction is defined as

$$E_p = \frac{A_p}{A}, \ \ p = o, g \tag{4.2}$$

where A_p is the area occupied by each phase and $A = A_o + A_g$.

An expression for the mixture velocity is obtained through the flux balance

$$\rho_m U_m = \rho_o E_o U_o + \rho_g E_g U_g \tag{4.3}$$

which yields the following mass conservative form of velocity

$$U_m = \frac{\rho_o}{\rho_m} U_{so} + \frac{\rho_g}{\rho_m} U_{sg} \tag{4.4}$$

where $U_{so} = E_o U_o$ and $U_{sg} = E_g U_g$, define, the respective liquid and gas superficial velocities.

The mass and momentum balance equations for the single pseudo-phase are expressed as

$$\frac{\partial}{\partial x}\left(\rho_m U_m A\right) = 0 \tag{4.5}$$

$$\frac{\partial}{\partial x}\left(\rho_m U_m^2 A\right) + \frac{\partial (Ap)}{\partial x} + \tau_{wm} S + \rho_m A g \sin\theta = 0 \tag{4.6}$$

where τ_{wm} is the wall friction shear stress and S is the wellbore perimeter. For constant pipe area *A*, combining Eqs. 4.5 and 4.6 yields

$$\frac{dp}{dx} = -\rho_m U_m \frac{\partial U_m}{\partial x} - \frac{\tau_{wm} S}{A} - \rho_m g \sin\theta \tag{4.7}$$

The right hand side terms are commonly known as the pressure drops due to acceleration, friction and elevation change, respectively. Brill and Beggs (1991) noted that the pressure drop due to acceleration is only significant if a compressible phase exists at relatively low pressures or the flowing area is changed. Since the coupled reservoir and facility models studied in this research are high-pressure systems and choke models are used to model flow through restrictions, the acceleration term is omitted.

The wall friction shear stress is computed from

$$\tau_{wm} = \frac{1}{2} f_m \rho_m U_m^2 \tag{4.8}$$

where f_m is the friction factor as computed by Jain (1976) and is a function of pipe roughness and Reynolds number

$$R_e = \frac{\rho_m U_m d}{\mu_m} \tag{4.9}$$

The mixture viscosity is defined to be

$$\mu_m = \mu_o E_o + \mu_g E_g \tag{4.10}$$

where μ_o and μ_g are functions of pressure P. The final form of Eq. 4.7 is given by

$$\frac{dp}{dx} = -\frac{2 f_m \rho_m U_m^2}{d} - \rho_m g \sin\theta \tag{4.11}$$

where d is the pipe diameter.

4.1.2 Slip Model

In uphill two-phase flow the gas phase has a tendency to travel at velocity higher than the liquid velocity and the velocity difference is commonly referred to as the slip velocity. The variations in phase velocities can have a direct impact on the pressure drop along the pipe. Even though the pseudo-phase approach treats the two phases as a mixture, slip between the phases can be incorporated via the liquid holdup calculation. In this research the method presented by Ouyang (1998) was used for computing liquid holdup. The relationship between the gas phase velocity and mixture velocity is based on slip parameters C_d and U_b defined by Wallis (1969) with

$$U_g = C_d U_m + U_b \tag{4.12}$$

C_d is a distribution coefficient related to velocity profile in the pipe and is determined empirically by

$$\begin{aligned} C_d &= 1.2 - 0.2\left(\frac{\rho_g}{\rho_o}\right)^{0.5} \\ U_b &= 1.53\left[\frac{g\left(\rho_o - \rho_g\right)\sigma}{\rho_o^2}\right]^{0.25} \sin\theta \end{aligned} \tag{4.13}$$

where U_b is the bubble rise velocity, and σ is the interfacial tension. The combined effect of these parameters is to allow for higher actual gas velocities as the difference in gas and liquid density increases. Given the modified gas velocity, new gas and liquid in-situ fractions can be calculated from

$$\begin{aligned} E_g &= \frac{U_{sg}}{C_d U_m + U_b} \\ E_o &= 1 - E_g \end{aligned} \tag{4.14}$$

The performance of this method is evaluated with an example in Section 4.1.4.

4.1.3 Calculation of Pressure Drop

To compute the pressure drop over a pipe interval from l_1 to l_2 with specified inlet conditions U_{so}, U_{sg}, E_o, E_g, and p_1, requires integration of Eq. 4.11

$$\int_{p_1}^{p_2} dp = -\int_{l_1}^{l_2} \frac{2U_m^2 f_m \rho_m}{d} + \rho_m g \sin\theta \; dl \tag{4.15}$$

Since the friction factor and density are functions of pressure an iterative process is required to compute p_2. Given a solution estimate p_2^{υ} at iteration υ, the average properties are computed with the following relations

$$\rho_m^{\upsilon} = \frac{1}{2}\left(\rho_{l_1} + \rho_{l_2}^{\upsilon}\right) \tag{4.16}$$

$$\mu_m^{\upsilon} = \frac{1}{2}\left(\mu_{l_1} + \mu_{l_2}^{\upsilon}\right) \tag{4.17}$$

and U_m is obtained from Eq. 4.4. The iteration equation to compute the downstream pressure is

$$p_2^{\upsilon+1} = p_1 - \left(\frac{2\left(U_m^2\right)^{\upsilon}}{d} f_m^{\upsilon} \rho_m^{\upsilon} + \rho_m^{\upsilon} g \sin\theta \right) \Delta l \tag{4.18}$$

where convergence is obtained when $\left| p_{l_2}^{\upsilon+1} - p_{l_2}^{\upsilon} \right| < \varepsilon$, for a specified tolerance ε.

In some cases a more accurate solution can be obtained through better property averaging by an increased number of iteration intervals along the pipe. Figure 4.1 shows how the number of integration intervals for two cases can affect the computed pressure drop. The first case simulates flow of oil at pressures above the bubble point throughout the entire pipe interval. Above the bubble point, gas solubility remains constant and oil formation volume factor increases slightly as pressure decreases. Eq. 3.5 shows that the liquid density is directly proportional to the oil formation volume factor and therefore density averaging over one integration interval is sufficient. The 50-interval result is

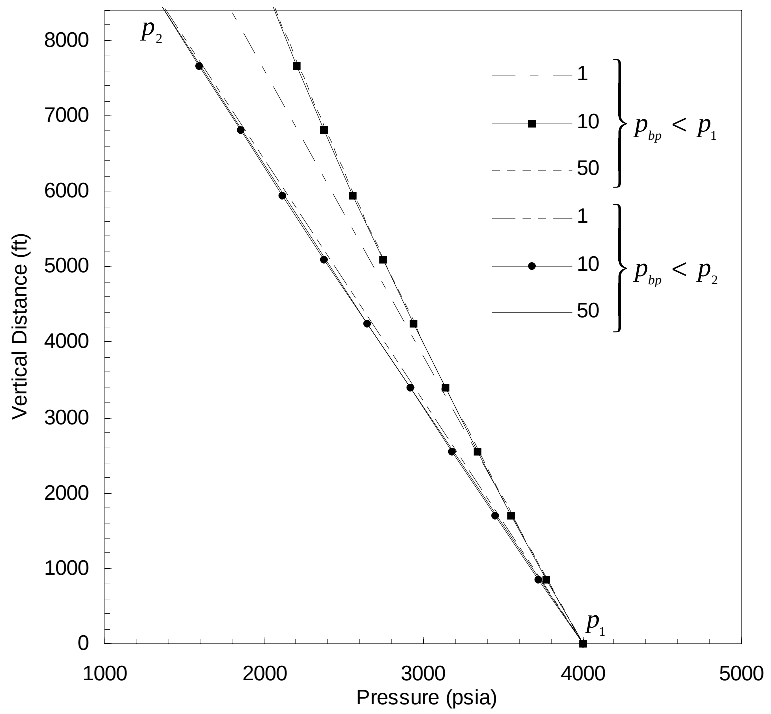


Figure 4.1: Pressure Drop Variation with Number of Integration Intervals

nearly identical to the 1- and 10-interval results. However in the second case the flow of oil occurs with pressures below the bubble point along the entire length of the pipe. In this regime oil density is a strong function of pressure and simple two-point averaging is not sufficient and resulted in a pressure error at p_2 of 14.35% with respect to the 50-interval result. In contrast, the 10-interval result resulted in an error of only 0.62% with respect to the 50-interval result.

4.1.4 Slip Model Example

Under certain conditions vertical two-phase flow exhibits the behavior of increasing flow rate with decreasing pressure drop. The slip model described in Section 4.1.2 is responsible for capturing this behavior. Figure 4.2 shows the simulated results based on vertical flow for both slip and no-slip cases, with uniform gas-oil ratio for all flow rates. Information regarding fluid properties and pipe parameters is summarized in Table 4.1.

Table 4.1: Slip Model Problem Data

Pipe Parameters		Fluid Properties	
diameter	3.5”	oil density	46.24 lbm/ft
roughness	0.0001’	gas density	0.0647 lbm/ft
vertical length	8400’	gas-oil ratio	500 scf/stb

The no-slip case has the behavior of increasing pressure drop with increasing rate for all ranges of rate since E_o and E_g are constant. However the case with slip shows that at low flow rates (fixed downstream pressure) the upstream pressure decreases with increasing flow rate. Since the liquid is denser, at lower flow rates the liquid has tendency to hold up in the pipe causing a large pressure drop due to the hydrostatic head. As the flow rate increases smaller pressure drops are required to sustain increased flow rates due to reduced holdup. As shown in Eq. 4.11 the pressure drop is composed of hydrostatic head and friction components. Figure 4.3 shows that for low flow rates the density component dominates the pressure drop (due to slip) and then stabilizes as the flow rate increases. At this stage the friction component increases and accounts for the

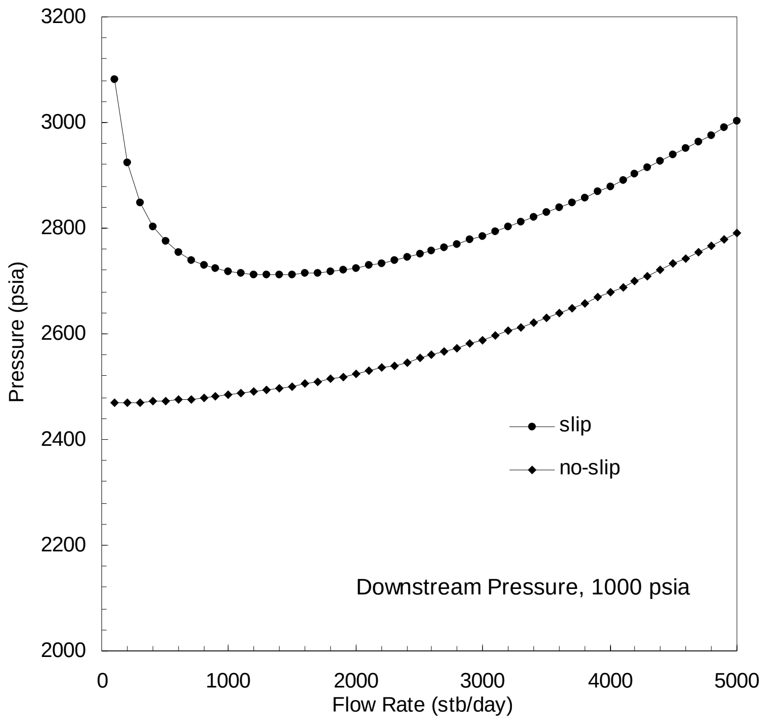


Figure 4.2: Comparison between Flow with Slip and No Slip

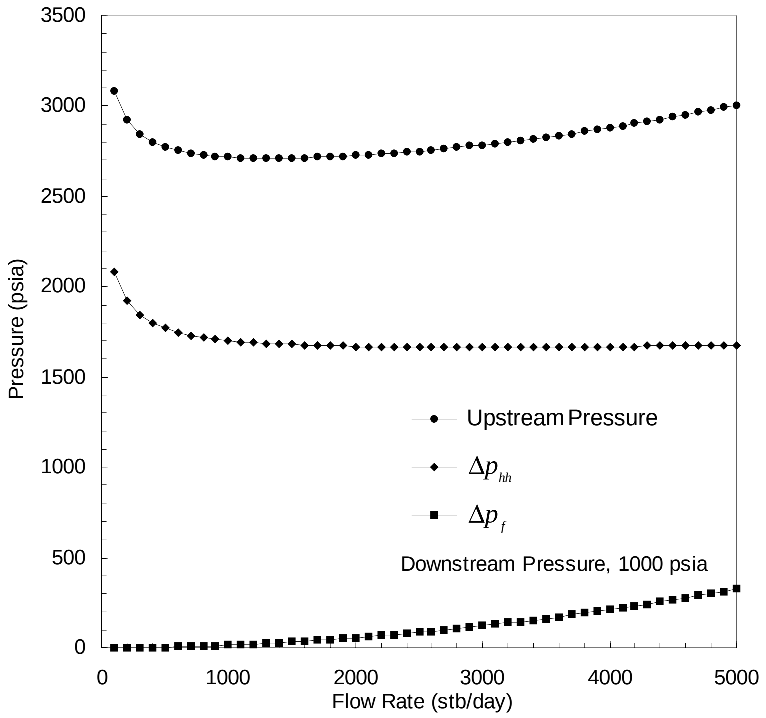


Figure 4.3: Component Pressure Drops in Flow with Slip

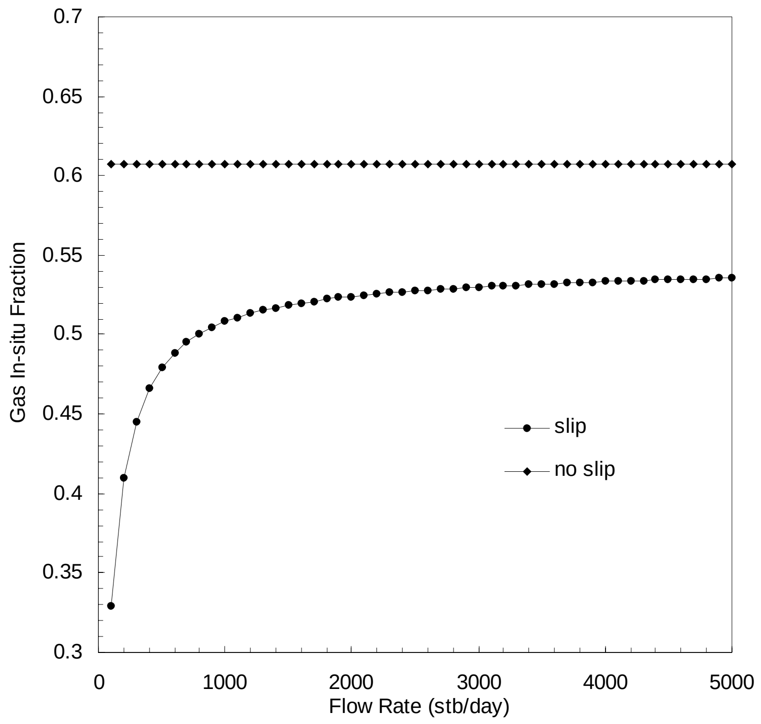


Figure 4.4: Gas Holdup Modification in Slip Model

increased pressure drop with increased rate. The mechanism in the slip model that drives the density behavior is seen in the gas holdup behavior shown in Figure 4.4. Since the gas liquid ratio was held constant, the no-slip case shows constant gas holdup. The slip case shows minor gas holdup at low rates, and translates into higher computed mixture density, thus accounting for the high pressure drop at low rates. As the flow rate increases the gas holdup increases, creating lower mixture density resulting in low pressure drops with increased rate.

4.2 Choke Model

A choke is used to regulate the total flow rate of a well or group of wells. The size of the choke determines the flow rate, which is classified as critical or subcritical flow. Critical flow implies that the fluid velocity through the choke is greater than the sonic velocity and that downstream pressure changes are not propagated upstream of the choke. Under subcritical flow the rate through the choke depends on the pressure differential across the choke. A good choke model incorporates both of these regimes and provides a smooth transition between the two types of flow. Based on the review of choke models presented by Schiozer (1994), the Sachdeva *et al.* (1986) choke model was implemented in this work. Both of these references provide a detailed mathematical formulation for the choke model and therefore this work presents only results that demonstrate the behavior of the model.

An example of how the liquid rate varies with upstream and downstream pressure is shown in Fig. 4.5 ($\rho_o = 51.24\,lb/ft^3$, $\gamma_g = 0.72$, and $GLR = 7000\ scf/stb$). The choke size is 16/64 inches, which also indicates that the choke is 75% closed. Since the choke is nearly closed, a large pressure drop is required to sustain even low flow rates. Note that there is a continuous transition between the critical and subcritical regions with the critical flow regions identified by the constant rate plateau.

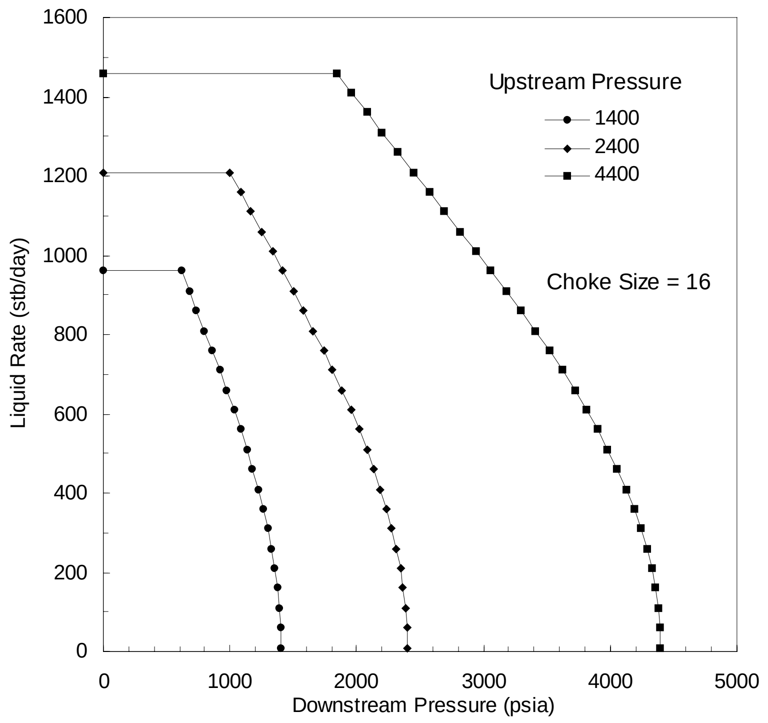


Figure 4.5: Critical and Subcritical Flow through a Choke

Wellhead Choke Performance

To properly evaluate choke performance, the well flowing characteristics must be evaluated over a range of reservoir pressures for a specified downstream pressure. The downstream pressure is determined by separation requirements and ideally the choke should operate under critical flow conditions so that pressure fluctuations in the surface facility equipment do not affect well performance, however, this is generally not the case. Fig. 4.6 ($\rho_o = 62.4\, lb/ft^3$, $\gamma_g = 0.72$, *absolute pipe roughness = 0.0001', vertical pipe distance 8400', pipe diameter = 3.5"*) shows the well flowing performance in terms of wellhead pressure over a range of well-block pressures. For a fixed well block pressure, the effect of slip causes the pressure drop in the tubing to decrease at low rates which corresponds to an increase in wellhead pressure. In the case of friction-dominanted flow, to sustain higher flow rates, the pressure drop must increase and therefore the wellhead pressure must decrease.

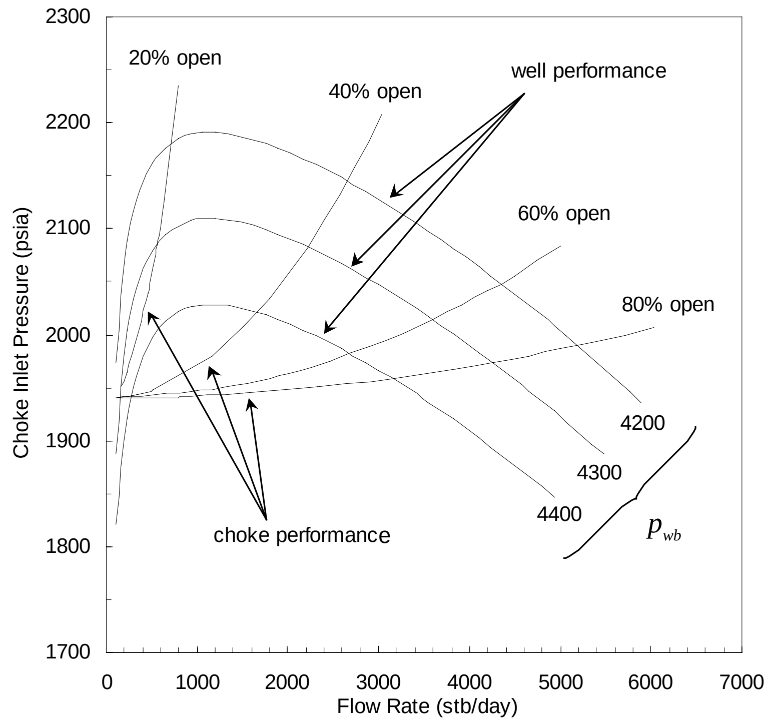


Figure 4.6: Wellhead Choke Performance

The well performance curves are overlaid with choke performance curves that correspond to a fixed downstream (wellhead) pressure of 1940 *psi* . The intersections of the well and choke curves represent consistent operating conditions at the tubing outlet and upstream side of the choke. As the choke is opened, the pressure drop across the choke decreases and the tubing head pressure approaches the fixed downstream pressure. As the choke is closed, the flow rate is reduced, and depending on the well block pressure, the well may cease to flow due to the high backpressure on the well. For example, if the well block pressure is 4400 *psi* and the choke is reduced from 40% to 20% open, the well and choke performance curves do not intersect and the well cannot operate. Beggs (1991) noted that the region of decreasing wellhead pressure with decreasing rate is unstable and can cause intermittent or no-flow conditions.

The main use of well and choke performance curves is to determine the choke setting which yields a target flow rate under changing reservoir conditions and specified downstream conditions. Traditionally, the intersection points that define the operating conditions are determined externally from the reservoir simulation model and are used to define reservoir boundary conditions in the form of well constraints. In this research a tightly coupled approach was developed in which the reservoir, well and choke models are formulated in a global problem, leading to a treatment of the model boundary conditions which are consistent with actual field operating conditions. The dependencies between the models are captured in the mathematical formulation thus avoiding the need for tubing and choke performance curves. The treatment of choke devices in the network model is presented in Section 4.4.2.

4.3 Separator Model

The collection of produced fluids requires phase separation prior to routing the flow stream to storage facilities or to sales pipelines. The size of the separator depends on the flow rate of liquids going into the vessel. The desired operating pressure and temperature can depend on the flow stream composition, well flowing pressures, or pressure of the gas sales line. An example of a multistage two-phase separation configuration is shown in

Fig. 4.7. In this schematic the operating pressure of the first stage separator defines the backpressure on the well flow stream. The total well stream flow rate is controlled by choke settings on individual wells. The operating pressure is maintained by pressure control valves on the gas line. As a consequence, the higher the operating pressure the smaller the compressor requirements for a gas sales line or injection system.

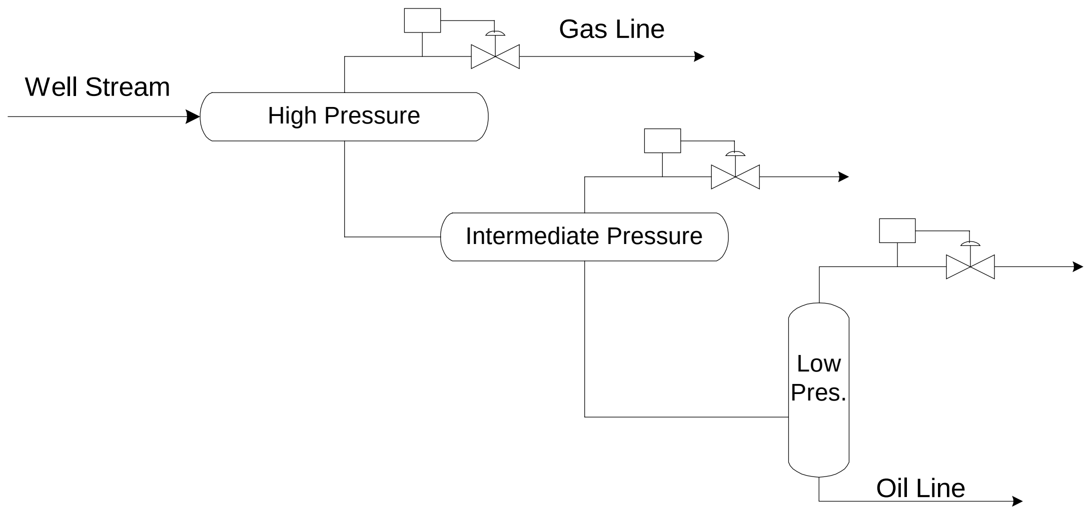


Figure 4.7: Stage Separation

In this research the fluids were characterized using a two hydrocarbon component formulation with limited phase behavior, therefore the oil-gas separation is based on a simple flash calculation using input fluid properties that define equilibrium conditions. Given a separator pressure and oil rate at standard conditions, the free gas phase is computed directly from the gas solubility. The treatment of a two-stage separator in a network model is presented in Section 4.4.3. It will become evident that the solution procedure for the global surface facility problem could also accommodate a more rigorous separation definition that allows for feedback between the stages and incorporates an equation of state for the equilibrium computations.

4.4 Network Model

The flow stream network currently used in this research follows the method first published by Startzman (1987). The network is defined by a collection of devices such as pipe, chokes, or separators. In addition, source and sink nodes are used to specify network boundary conditions and connection nodes are used to join devices. A conceptual network model is employed to facilitate discussion (Fig. 4.8). Three wells are connected to a common junction node that leads into the separation unit. The separator pressure and well flowing rate or pressure defines the sink and source boundary conditions, respectively. Connection nodes join the well tubing with the surface pipeline and also define a junction point for combining the produced fluids. In this example it is assumed that computed well rates or pressures are consistent with the reservoir deliverability. The next chapter provides a detailed treatment of reservoir, well and facility coupling.

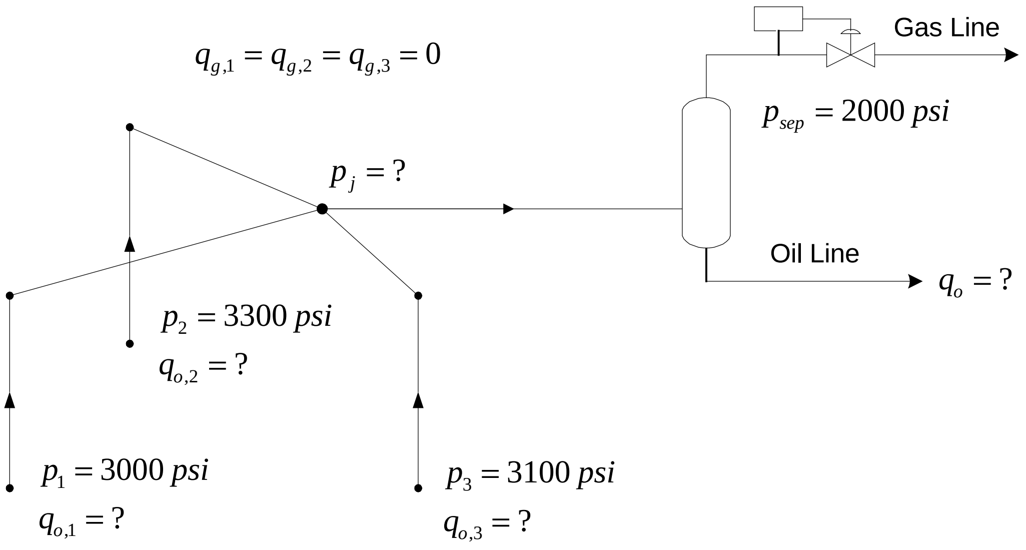


Figure 4.8: Conceptual Network Layout

The flexibility in defining network connectivity with a different set of devices allows for modeling a variety of operational scenarios. For the model shown in Fig. 4.8

the objective is for a specified well tubing and pipeline configuration, determine well rate allocation when the gathering system is operating under a constant separator pressure. If individual well rate restrictions apply, or a target group oil rate is specified, wellhead choke devices are required and the objective would become to determine the choke settings which honor these rate conditions. An example of this type of configuration is presented in the next chapter.

4.4.1 Network Solution Procedure

The solution procedure is discussed with respect to a network consisting of only pipe equipment. The treatment of choke and separator devices is presented in the following section. Two test models are presented in Section 4.5 to demonstrate network features and convergence behavior.

Mass Conservation

The network problem is formulated in terms of continuity relationships at source, sink and junction nodes. Based on steady state flow and the single pseudo-phase treatment of pipe flow, a total mass conservation equation at each node is expressed as

$$\sum_{j\in\Omega_i}\left(\rho_o U_{so} A + \rho_g U_{sg} A\right)_{ij} = 0,\ i = 1,..,n_f \tag{4.19}$$

where Ω_i is the set of all nodes connected to node *i* and n_f is the total number of facility nodes. The velocity is defined to be positive if the flow direction is from node *j* to *i*, otherwise the velocity is negative. Similar to component mass conservation equations developed for the reservoir problem, Eq. 4.19 can be expressed as a system of component equations in terms of mass fractions to obtain

$$\begin{aligned}&\sum_{j\in\Omega_i}\left(\rho_o x_{oo} U_{so} A\right)_{ij} = 0,\ x_{og} = 0\\&\sum_{j\in\Omega_i}\left(\rho_o x_{go} U_{so} A + \rho_g x_{gg} U_{sg} A\right)_{ij} = 0\end{aligned} \tag{4.20}$$

Total mass is conserved in each individual pipe segment via Eq. 4.5. In the production gathering system only the oil component equation is formulated for Newton's method. The following residual form of Eq. 4.20 is used to formulate the conservation equations for the production network system:

$$r_{o,i} = \sum_{j \in \Omega_i} \left(\rho_o x_{oo} U_{so} A \right)_{ij} - M_{oo,i} \tag{4.21}$$

where M_{oo} is the oil mass flow rate for a source term at node *i*. The gas phase due to boundary conditions or evolution of solution gas is accounted for at a junction node by ensuring that the sum of all free gas flowing into the junction node is exactly the amount of gas flowing out of the junction node. Subsequent Jacobian or residual calculations for this pipe segment cannot be performed until computations are completed for all pipe segments with flow into the junction node, and thus an order dependency exists for constructing the Jacobian and residual terms.

The injection network only allows for single-phase single component gas injection so that $x_{oo} = x_{go} = 0$ and $x_{gg} = 1$. Therefore the following residual form of Eq. 4.20 is used to formulate the conservation equations for the injection network problem:

$$r_{g,i} = \sum_{j \in \Omega_i} \left(\rho_g U_{sg} A \right)_{ij} - M_{gg,i} \tag{4.22}$$

where M_{gg} is the gas mass flow rate for a source term at node *i*.

Newton's Method

The system of equations is solved by Newton's method. Assuming a production gathering system, a Taylor series expansion of Eq. 4.21 yields

$$r_{o,i}^{\upsilon+1} = r_{o,i}^{\upsilon} + \sum_{j \in \Omega_i} \frac{\partial r_{o,i}}{\partial p_j} \Delta p_j + \frac{\partial r_{o,i}}{\partial p_i} \Delta p_i + \frac{\partial r_{o,i}}{\partial M_{oo,i}} \Delta M_{oo,i} \tag{4.23}$$

so that the linear system can be expressed compactly as

$$\begin{aligned} A_f^{\upsilon}\delta\vec{X} &= -\vec{R}_f^{\upsilon} \\ \delta\vec{X}_f &= \vec{X}_f^{\upsilon+1} - \vec{X}_f^{\upsilon} \end{aligned} \tag{4.24}$$

where A_f^{υ} and R_f^{υ} are the respective Jacobian matrix and residual vector.

Formation of the Jacobian matrix is via numerical computation of the derivative terms for either Eq. 4.21 and/or 4.22. The numerical method is presented for a junction node in a production gathering system. Assuming no sources or sinks Eq. 4.23 can be expressed as

$$-r_{o,i}^{v} = \sum_{j\in\Omega_i} \frac{\partial q_{o,ij}}{\partial p_j}\Delta p_j + \frac{\partial q_{o,ij}}{\partial p_i}\Delta p_i \tag{4.25}$$

where $q_o = \rho_o x_{oo} U_{so} A$. Considering only the first term on the right hand side, the summation represents the derivative of the flow rate between node *i* and *j* with respect to a pressure at node *j*. For each of the *i-j* connections the following procedure is applied:

- a small perturbation (δq_o) of the current rate is applied to the *i-j* connection
- the perturbed rate, current pressure estimate, and total free gas phase at *j* are used to compute a new downstream pressure via Eq. 4.18, yielding

$$\delta p_j = p_j^{new} - p_j^{current} \tag{4.26}$$

- the derivative is then given by

$$\frac{\partial q_{o,ij}}{\partial p_j} \cong \frac{\delta q_o}{\delta p_j} \tag{4.27}$$

The current rate and associated downstream pressure are available from either a specified initial estimate or the most recent network solution.

Formation of the initial residual vector, $\vec{R}_f^0$, is based on a combination of specified estimates for the source nodes and computed estimates for the remaining nodes. For example, in the network shown in Fig. 4.8, assuming oil rate estimates $q_{o,i}^0$ are provided $(\upsilon = 0)$, the following procedure is used to determine other node estimates and residuals:

- for each well i with rate estimate $q_{o,i}^0$, boundary conditions p_i and $q_{g,i}$, a corresponding junction node pressure $p_{j,i}$ is computed
- the starting solution for the junction node pressure p_j, is computed from the average of computed pressures, $p_{j,i}$
- the junction node pressure p_j and specified separator pressure p_{sep} are used to compute the rate estimate $q_{o,3}^0$

where pressure and rate computations rely on Eq. 4.18 for either computing the downstream pressure directly, or computing the rate indirectly using a Newton-Raphson scheme. Subsequent residual vectors are computed in an analogous manner with the exception that all current estimates are from the Newton iteration update.

The relationship between pressure drop and velocity is apparent from Eq. 4.18 and is expressed as

$$U_m \, \alpha \sqrt{\Delta P} \tag{4.28}$$

so the derivative terms in Eq. 4.29 will have the form

$$\frac{\partial U_m}{\partial p_i} \, \alpha \frac{1}{2\sqrt{\Delta p}} \tag{4.29}$$

The nonlinear relationship between velocity and pressure can lead to slow convergence rates. To control problems associated with poor derivative information, rules that define the maximum change in the primary variable over the iteration, are imposed by the condition

$$x_i^{\upsilon+1} = \min\left\{x_i^{\upsilon} + \Delta x_i^{\upsilon}, x_i^{\upsilon}\left(1 \pm \eta\right)\right\} \tag{4.30}$$

where $\eta \cdot 100$ is the maximum percent change in the previous solution value, x_i^ν. The effect of different step scaling factors on convergence rate is demonstrated in the Section 4.5.

4.4.2 Choke Implementation

Individual well or group flow rate restrictions may require adjustment of the choke settings to ensure that these rates are not violated. The choke model described in Section 4.2 is incorporated with the network model to allow for either fixed or variable choke settings. For a fixed choke size, the presence of a choke in the network model is treated in a similar fashion to that of the pipe device, it is bounded by two solution nodes and the mathematical model describing flow through the device is used to compute numerical derivatives. When the objective is to determine the choke setting corresponding to a fixed flow rate, the setting is modified within the Newton iteration and can be viewed as restarting the network solution procedure with an improved starting solution. After every

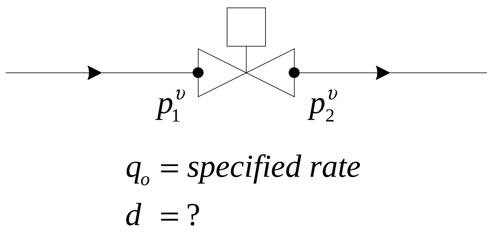


Figure 4.9: Variable Choke for Operating at Constant Rate

Newton iteration, the new estimate for the choke setting is computed based on the specified rate, and current pressures p_1^ν and p_2^ν (see Fig. 4.9) using the choke equation. If the estimate exceeds the diameter corresponding to 100% open, the choke is set to fully open. The next chapter presents an example that demonstrates choke operation under changing reservoir conditions and a fixed downstream pressure boundary condition.

4.4.3 Separator Implementation

The specification of separation devices in the network model is similar to defining boundary conditions for the downstream and upstream portions of the production gathering and injection system, respectively. The treatment of this device in the Newton solution procedure is described in the context of a two-stage separator that provides input for the injection network (Fig. 4.10). For simplicity, it is assumed that there is no pressure drop through oil or gas lines connecting the separators. Also, it is assumed that the pressure regulator valve is set properly to maintain the separator pressure and therefore pipeflow and valve models are not applied to the gas sales line. The internal representation of the system shown in Fig. 4.10 is outlined as follows:

- The first stage separator defines a Dirichlet boundary condition for the production gathering network, $p_{sep1} = 2000\ psia$. The unknown at this node is the oil rate. From the current oil rate solution, q_o^υ, the free gas q_{g1}^υ is computed using gas solubility tables.
- The oil rate q_o^υ, is used in the flash calculation at the second stage separator conditions, $p_{sep1} = 1500\ psia$, to obtain q_{g2}^υ.
- If the available gas is used for reinjection, a Neumann boundary condition is specified for the source node of the injection network, with the gas rate based on the previous Newton iteration solution.
- If the objective of the injection network is for pressure maintenance, the source node requires a Dirichlet boundary condition and the unknown is the gas rate. In this case, there is no coupling in the formulation between the separation units and source node for the injection network. The available free gas from the separation units is used to determine the deficit or surplus gas amount, externally.

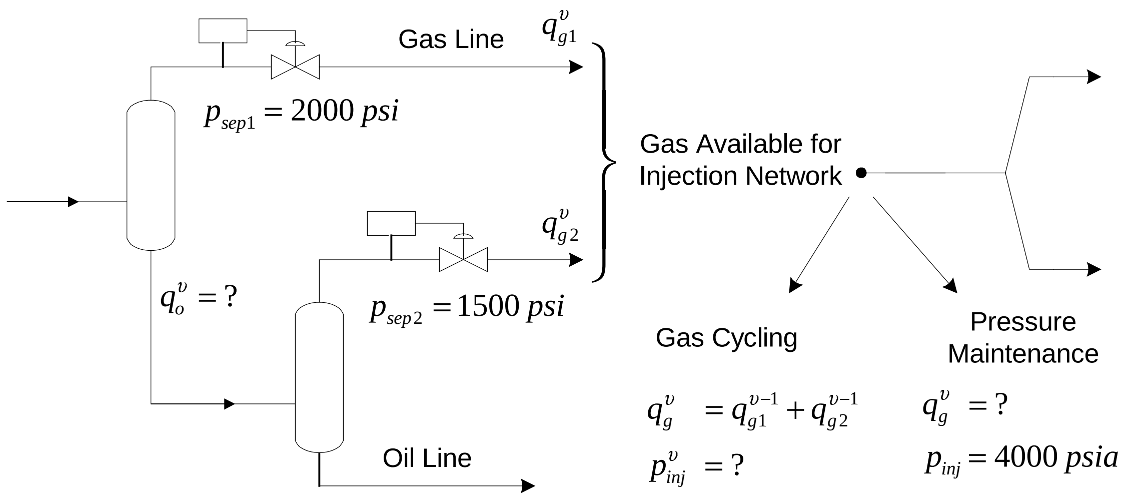


Figure 4.10: Two-Stage Separation for Input to Injection Network

4.5 Model Examples

In this section, results from two surface facility models will be presented and discussed. Both test models demonstrate typical convergence behavior that can be expected for Newton's method. The first model represents a production gathering system and the second model includes a well constraint and a gas injection network. The model parameters are listed in Table 4.2 and are the same for all models unless otherwise specified.

Since the intent of this research is to study coupled reservoir and facility models, the surface facility program was not designed to run without reservoir coupling. However, the behavior of the facility model can be studied in isolation by replacing a Dirichlet boundary condition at a wellbore source or sink node with a fixed well block pressure. Specification of this type of boundary condition is used for explicit treatment of the facility model in coupled systems and is described in detail in the following chapter.

Table 4.2: Basic Network Parameters

Pipe Parameters	Diameter	Roughness	Horizontal Length	Vertical Depth
well tubing	3.5"	0.0001'	0	8400'
wellhead to junction	3.5"	0.0001'	6363'	0
junction to separator	5.5"	0.0001'	4000'	0

Choke Parameters	Diameter	Discharge Coefficient
PROD1, C_1	0.8"	0.85
PROD2, C_2	0.5"	0.85

4.5.1 Two Production Wells Operating Under Fixed Choke Settings

In this example two wells are flowing at constant well block pressure (p_{r1}, p_{r2}) with fixed choke settings at the wellhead (Fig. 4.11). The well flow streams are combined and feed into a separator operating at constant conditions. The pipeline layout is designed so that in the absence of chokes, each well will have identical flowing rates and flow line pressure drops. The addition of chokes with different settings resulted in different flow rates for each well. As shown in Fig. 4.6, for a constant downstream pressure a reduction in the choke size has the affect of increasing the backpressure on the well and reducing the flow rate. The rate allocation shown in Fig. 4.12 is consistent with this behavior. The chokes for wells *PROD1* and *PROD2* are set at 80% and 50% open, thus well *PROD1* correspondingly has the higher flow rate.

Four test runs were performed using the two starting solutions $(k = 1, 2)$ shown in Table 4.3. The starting solution for *run 1* is considerably better than for *run 2*. The second run estimates are an extreme case to highlight problems associated with poor solution estimates. The estimates used for the third and fourth runs are identical to those of the second run. Additionally, the third and fourth runs use the modified Newton strategy presented Section 4.4.1 to control the maximum primary variable change over an iteration. For each of the runs, convergence behavior at the solution nodes is shown in Figs. 4.12 through 4.15. The second subscript in the figure variables indicates which starting solution was used to generate the results.

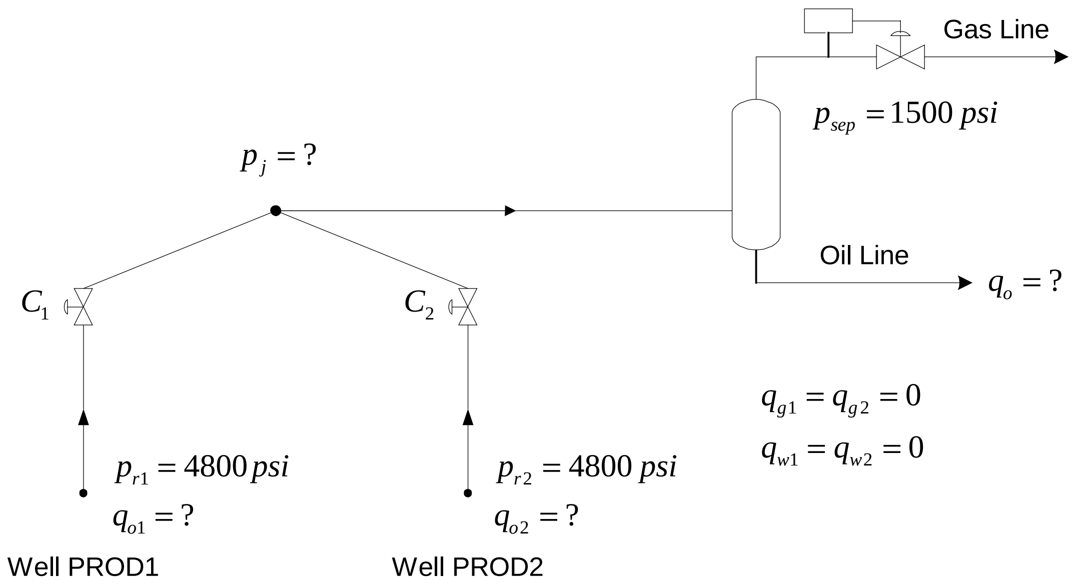


Figure 4.11: Network Example with Two Production Wells – Model 1

Table 4.3: Network Starting Solution – Model 1

Starting Solution, $k = 1,2 \quad \upsilon = 0$	$k = 1$	% Error	$k = 2$	% Error
PROD1 $q^{\upsilon}_{o1,k}$ (stb/day)	8000	12.75	4300	53.10
PROD2 $q^{\upsilon}_{o2,k}$ (stb/day)	7000	5.87	3000	54.63
Junction $p^{\upsilon}_{j,k}$ $(psia)$	1642	4.39	2435	54.80
Separator $q^{\upsilon}_{sep,k}$ (stb/day)	25,572	62.03	75,689	379.59

Runs 1 and 2

The first run shows a good Newton convergence rate for all the solution nodes and the global problem converged within five iterations. However, *run 2* shows a much slower convergence rate. Recall that initial source node estimates are used to compute starting solutions for nodes downstream of well nodes, which in this model involves the junction and sink nodes. For *run 2* the computed junction node pressure allowed for flow into the separator (Fig. 4.14). However, after the first Newton iteration, the junction node pressure update created a backflow situation at the separator. Examination of the junction and separator pressures in Table 4.3 shows that a very high pressure drop exists over the flow line into the separator. This translates into a junction node residual based on a large exit flow rate (*exceeds solution by ~400%*) and smaller inflow rates from the well estimates. In an effort to meet the high flow rate into the separator, the first Newton step will lower the junction pressure to obtain higher rates from the wells. In this case, the pressure correction generated an infeasible solution. At this stage the algorithm will automatically set the flow line into the separator to a no-flow condition so that the junction node residual only has positive contributions from the wells. Since there is no flow exiting the node, the Newton procedure is forced to increase the pressure at the junction node. This continues until the junction node pressure is higher than the separator pressure, which in this run required nine iterations. Once this state has been achieved the Newton iteration converges rapidly. In total, *run 2* required 12 iterations for global convergence, considerably more than *run 1,* which required only 5 iterations.

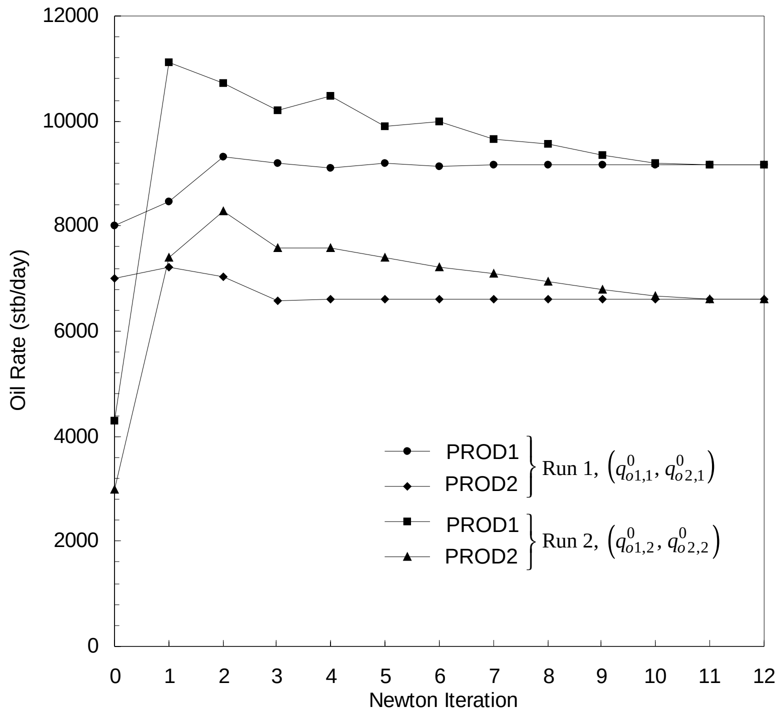


Figure 4.12: Newton Convergence Rate without Scaling Factor – Prod. Wells

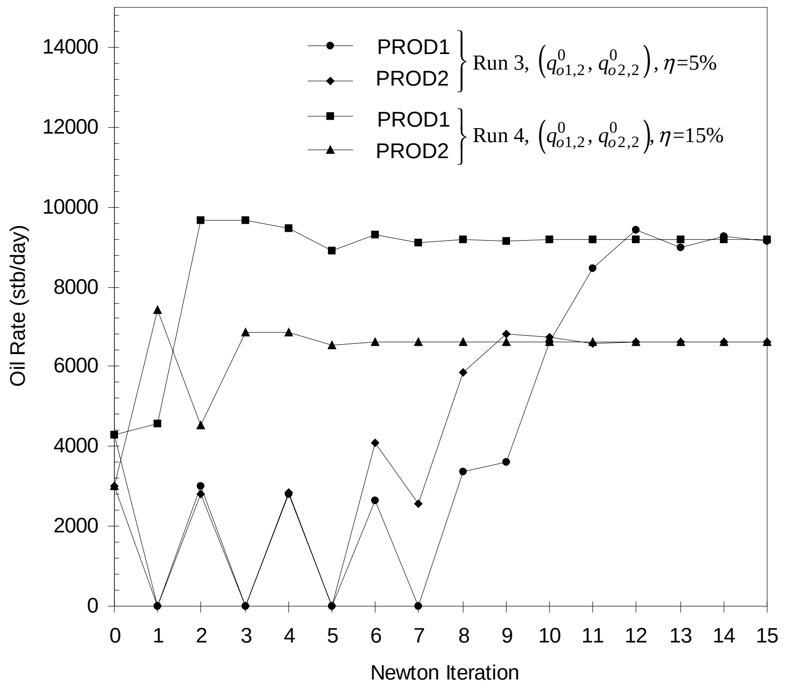


Figure 4.13: Newton Convergence Rate with Scaling Factor η – Prod. Wells

Runs 3 and 4

The slower convergence rate associated with *run 2* is due to the computed solution estimate for the junction node, which created a large pressure gradient over a horizontal line and ultimately required a large pressure correction. To avoid this situation, the third and fourth runs use the scaling criteria defined in Eq. 4.30, which restricts the maximum change in the junction node pressure over an iteration. In these runs, the maximum allowable change in the updated pressure is specified as 5% and 15% of the previous solution, respectively. In both cases, the effect of the scaling is quite evident in Fig. 4.14, which shows the pressure solution changing at a constant rate. A similar profile is shown in the rate solution for the separator (Fig. 4.15).

While both levels of scaling have successfully prevented the backflow situation, the more restrictive scaling used in *run 3* has introduced other problems. In this run, the well rates oscillate between a flow and no-flow state during the initial iterations. The cause can be understood by considering the pressure behavior of the junction node and chokes (Figs. 4.14 and 4.16). The scaling of the junction pressure has slowed the convergence of the pressure solution at the junction, and at the choke outlet. This results in several iterations, which generate a high back pressure on the wells, and thus accounts for the no-flow behavior seen in Fig. 4.13.

Recall that in *run 2* where due to the back flow, the Newton procedure was forced to first obtain a better solution upstream for well rates, which then allowed for convergence in the downstream nodes. In contrast due to scaling, *run 3* resulted in the Newton procedure first improving the solution downstream at the junction and separator nodes, before addressing the rate allocation. This is illustrated in Fig 4.16, which shows no improvement in the upstream side of the choke until the downstream side has moved closer to the solution. The convergence rate for *run 3* was comparable to *run 2* and required 15 iterations for global convergence.

The fourth run used a less restrictive scaling in response to the performance observed in *run 3* and resulted in an accelerated correction rate of the starting solution estimates, especially at the junction node. Throughout the iterations, no convergence

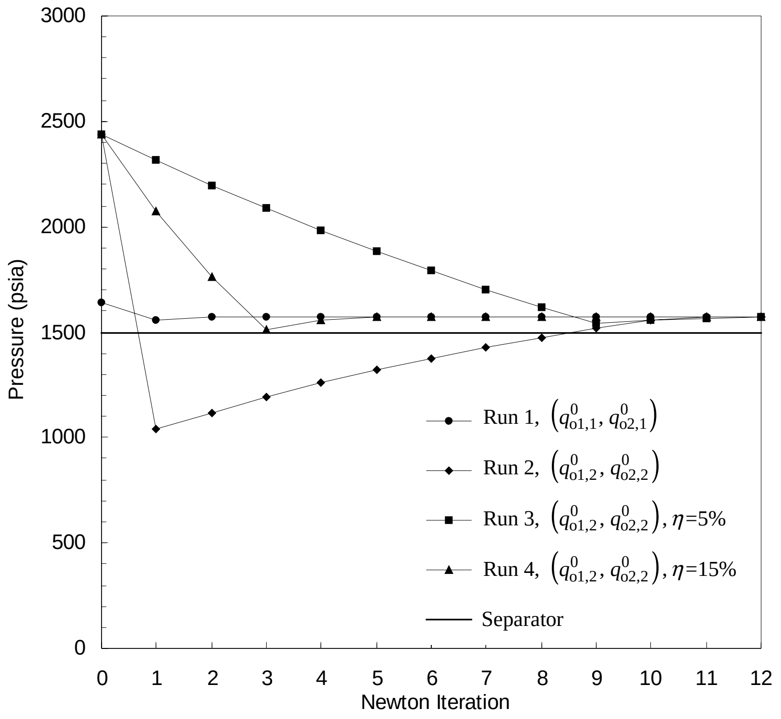


Figure 4.14: Newton Convergence Behavior for Junction Node

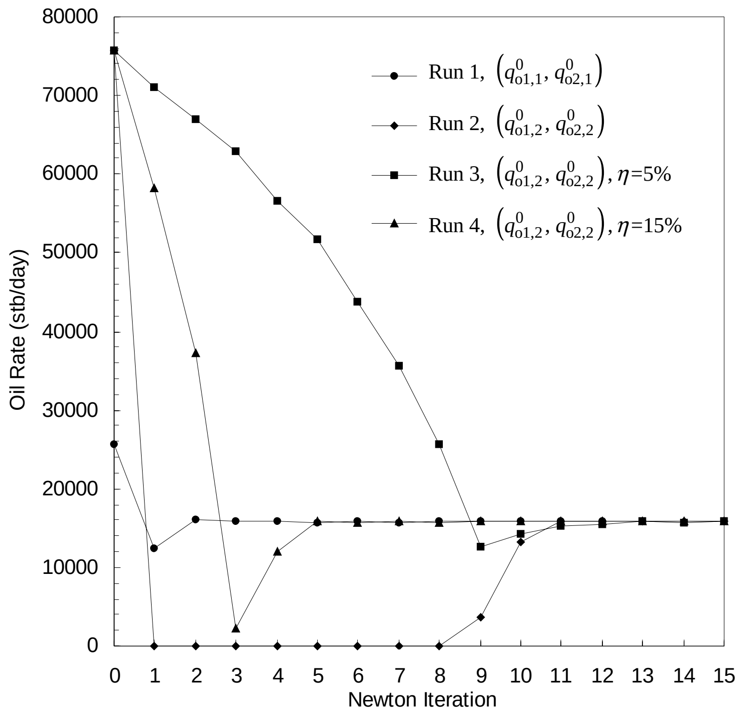


Figure 4.15: Newton Convergence Behavior for Sink Node

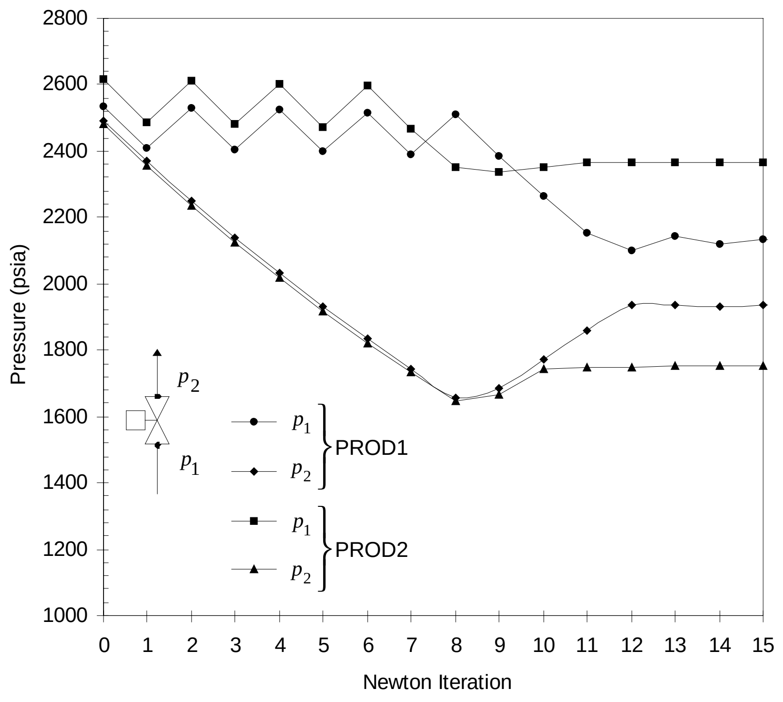


Figure 4.16: Newton Convergence Behavior at Chokes – Run 3

problems were encountered due to no-flow situations in either the well, or downstream into the separator. In this run, the convergence rate compared very favorable with the *run 1*, and required only six iterations for global convergence.

4.5.2 Well Constraint and Gas Cycling

The second test example was used to demonstrate the ability of the formulation to incorporate a well rate constraint into the model via the procedure described in Section 4.4.2. Additionally, two injection wells are included to simulate a gas cycling operation that also has gas available from an external source as shown in Fig. 4.17. The new network parameters and boundary conditions are shown in Tables 4.4 and 4.5, respectively. Except for the variable choke assigned to *PROD1*, the production network parameters are the same as in the previous example.

Two runs were performed to test the sensitivity of the starting solution with respect to choke diameters. Starting solutions for the model are listed in Table 4.6. Similar to the previous model, two additional runs were performed to investigate the impact of scaling the pressure solution. Due to the similarity in results, only the runs with a scaling value of $\eta = 15\%$ are presented. References to *run 1* and *run 2* indicate results based on different starting solutions, but application of the same scaling value. Where appropriate, comments on the other runs are included (no scaling or $\eta = 5\%$).

Table 4.4: Injection Network Parameters – Model 2

Pipe Parameters	Diameter	Roughness	Horizontal Length	Vertical Depth
gas source to junction	6.0”	0.0001’	4000’	0
junction to wellhead	6.0”	0.0001’	6362’	0
well tubing	2.5”	0.0001’	0	8400’

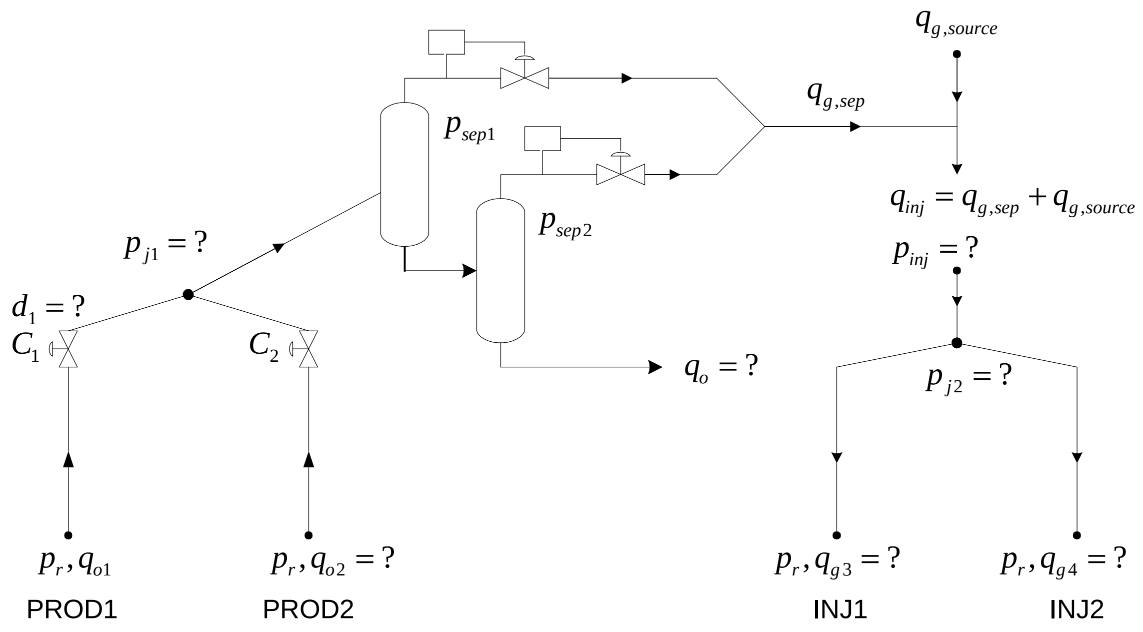


Figure 4.17: Facility Example with Production and Injection Network – Model 2

Table 4.5: Injection Network Boundary Conditions – Model 2

Dirichlet		(psia)	Neumann		
Well block	p_r	4800	Gas Source	$q_{g,source}$ $(MMscf\ /\ day)$	40
1st Stage Separator	p_{sep1}	1500	Gas Injection	q_{inj} $(MMscf\ /\ day)$	$q^{\upsilon-1}_{g,sep} + q_{g,source}$
2nd Stage Separator	p_{sep2}	500	PROD1	q_{o1} $(stb\ /\ day)$	8000

Table 4.6: Network Starting Solution – Model 2

Starting Solution, $k = 1, 2 \quad \upsilon = 0$		$k = 1$	% Error	$k = 2$	% Error
PROD1 Choke	C_1 $(in.)$	0.95	55.73	0.45	26.23
PROD2	$q^{\upsilon}_{o2,k}$ $(stb\ /\ day)$	5000	24.82	5000	24.82
Junction 1	$p^{\upsilon}_{j1,k}$ $(psia)$	1967	26.00	1351	13.45
Separator	$q^{\upsilon}_{sep,k}$ $(stb\ /\ day)$	52126	255	0	100
Gas Source	$q^{\upsilon}_{inj,k}$ $(MMscf\ /\ day)$	34.46	0.40	33.94	1.89
Junction 2	$p^{\upsilon}_{j2,k}$ $(psia)$	6991	21.98	6991	21.98
INJ1	$q^{\upsilon}_{g3,k}$ $(MMscf\ /\ day)$	17.23	0.40	16.97	1.89
INJ2	$q^{\upsilon}_{g4,k}$ $(MMscf\ /\ day)$	17.23	0.40	16.97	1.89

The choke diameter estimates were selected to reflect two different starting solution strategies, beginning with a well flowing 1) at full potential and 2) at a considerably restricted rate. In this model these estimates correspond to a choke setting of $d_{1,1}^{0} = 0.95''$ *(run 1)* and $d_{1,2}^{0} = 0.45''$ *(run 2)*, respectively. As shown in Fig. 4.18, the algorithm was able to converge on the solution within 12 Newton iterations for either starting solution. The cases without scaling and with a *5%* scaling showed similar convergence behavior. Recall, that internally a choke is represented by two pressure solution nodes placed at the choke inlet and outlet. Examination of the convergence behavior at these nodes shows the affect of scaling in *run 2* (linear change in pressure p_2 at choke outlet, Fig. 4.19). Also note in this figure that compared to *run 1*, the smaller choke size estimate used in *run 2* resulted in a much larger starting pressure drop across the choke. Similar to the previous model, the source of the scaling can be found by examining the pressure at the junction node (Fig. 4.20). The computed solution estimate for this node is below the downstream separation pressure. For the specified flow rate boundary condition, the choke C_1 set at 55% closed creates a large pressure drop across the choke that subsequently reflects in a lower starting junction pressure estimate. Therefore, for *run 2* the starting solution includes a no-flow situation with respect to flow into the separator. The unscaled and 5% scaling runs based on the *run 2* starting solution also show similar convergence behavior due to the strong impact of the poor choke setting estimate.

As shown in the network diagram, the separated gas is combined with a constant external gas supply for reinjection into two wells. This relationship between the production network solution and injection network boundary condition couples their convergence rates. However, for most of the Newton iterations the total gas injected was close to the source amount (Fig. 4.21) and indicate a relatively good pressure solution. Since identical flow line configurations were used for each injection well, the well gas rate allocation was identical and the rate profiles follow that of the total injection rate.

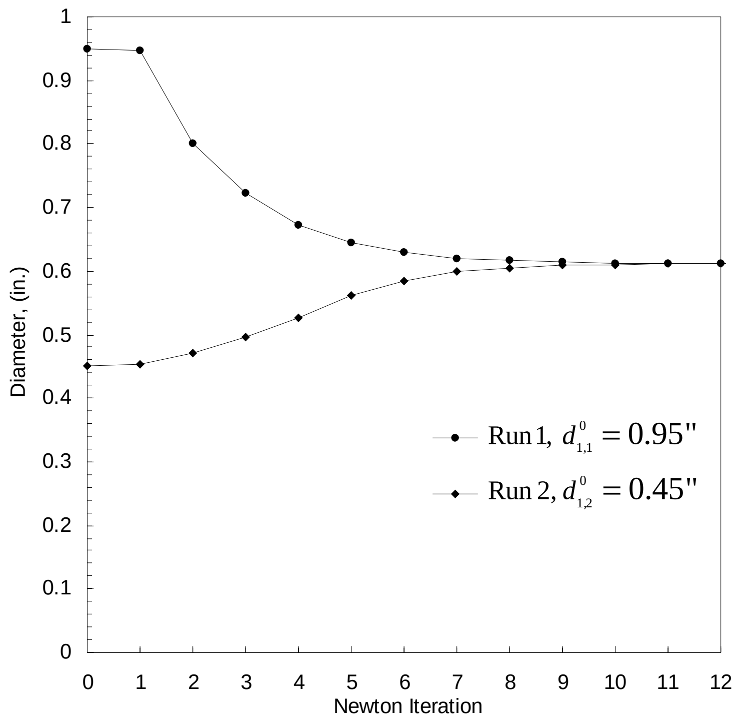


Figure 4.18: Newton Convergence Behavior for Choke Diameter

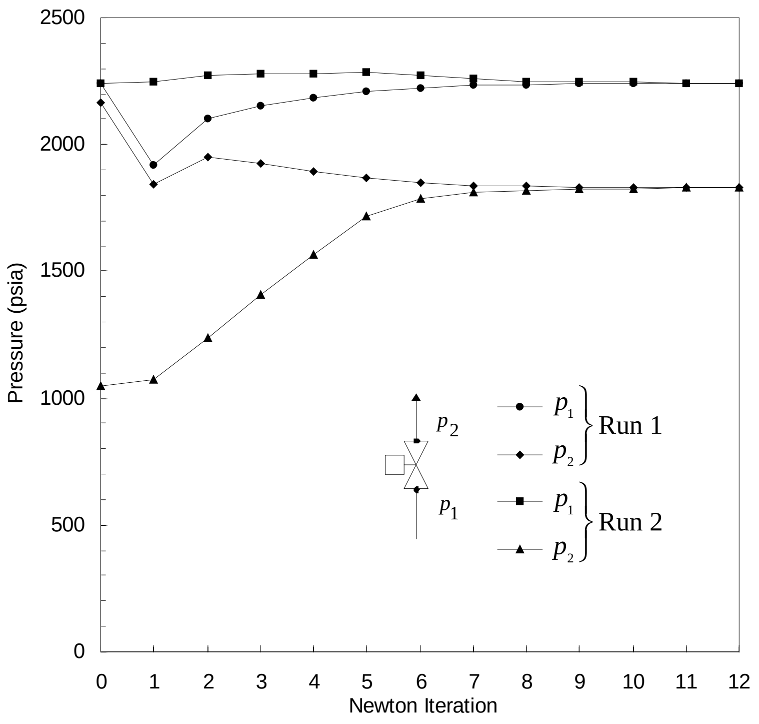


Figure 4.19: Newton Convergence Behavior, Pressure Drop Across Choke

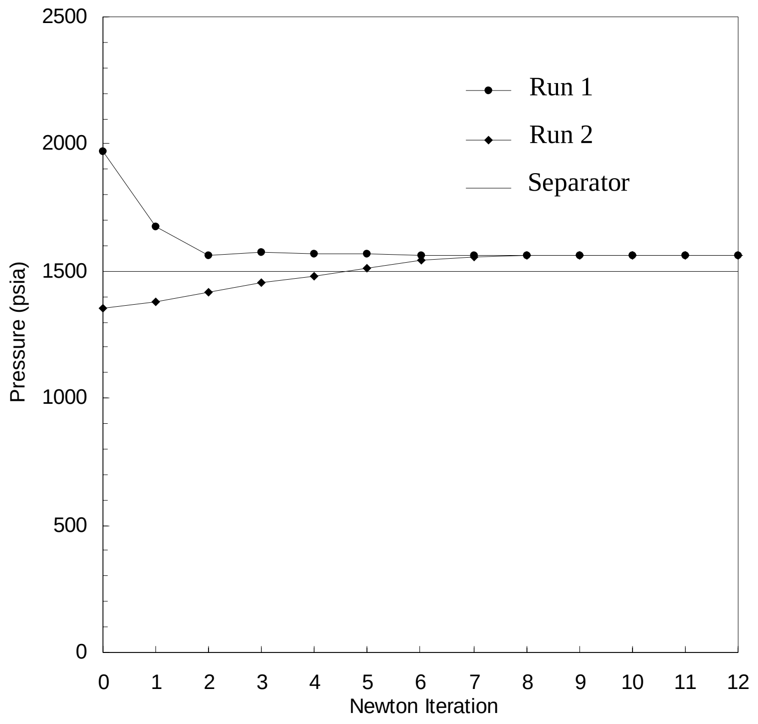


Figure 4.20: Newton Convergence Behavior at Injection Network Junction Node

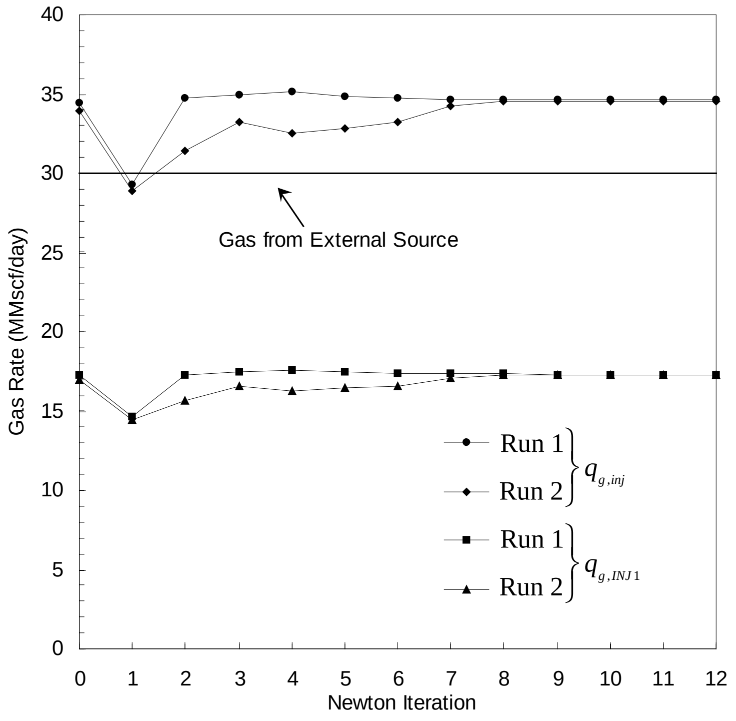


Figure 4.21: Newton Convergence Behavior at Gas Injection Node

4.6 Concluding Remarks

The device models that are used to construct a surface facility model have been presented and individual performance characteristics discussed. The test results for the network models have highlighted the type of convergence behavior that can be expected from a surface facility model. The quality of the starting solution has a direct impact on the convergence rate and for poor quality estimates, scaling of the solution may improve the convergence rate. However, the scaling factor is problem dependent. More importantly, the robustness required in the solution procedure is evident from equipment and feasibility problems associated with poor initial solutions, solution scaling, or Newton step updates.

The convergence behavior was analyzed with respect to problems associated with flow imbalances or pressure estimates at the junction node. The simplicity of the two network models may lead to obvious methods to accelerate convergence. For example, rules could be specified to restrict the maximum inflow and outflow imbalance in an attempt to locally minimize the correction from the Newton step. However, in a complex network configuration, where there may be several internal junction nodes, the use of local rules are of limited use since resolution of a local problem may introduce a problem in another part of the facility model. More complicated (rule) schemes could be devised that are more global in scope, but in the limit they are competing with updates based on the Newton iteration, which contains global dependence information via the Jacobian. Therefore we have chosen to use the basic Newton method with simple rules for variable scaling and treatment of no-flow situations, and forgo the implementation of more complex rules which: 1) would considerably increase the coding effort and 2) do not guarantee improvement in the rate of convergence. Test models that demonstrate the impact of the network convergence behavior on a fully coupled surface facility and reservoir model are presented in the next chapter.

Chapter 5

Coupled Reservoir and Surface Facility Models

Methods for coupling reservoir and surface facility models may be based on either implicit or explicit formulations. The specific formulation depends on the placement of boundary conditions and the components within the Newton iteration. This research examined both explicit and implicit formulations. Adaptive explicit facilities were developed that combine the best properties of the explicit and implicit formulations. This chapter presents the implementation details used in the formulations. A new preconditioning technique for the fully implicit, adaptive implicit, and adaptive explicit facility methods is presented. The new methods are contrasted with current techniques for coupled reservoir and facility models and are shown to provide a significant improvement in computational efficiency.

5.1 Explicit Facility Coupling

The explicit formulation considered in this research is based on decomposing the model into reservoir and facility domains with domain interfaces placed in the neighborhood of well blocks. The facility problem is formulated using boundary conditions based on the previous timestep reservoir conditions. The problem is solved using Newton's method subject to Dirichlet or Neumann boundary conditions that are imposed via the well equation for the reservoir model. The rate or pressure constraints are consistent with the facility boundary conditions and the reservoir deliverability at the end of the previous timestep, but are applied to the reservoir model for the current timestep. The following example is used to further describe the method.

Consider the coupled system shown in Fig. 5.1 where the production network is connected to a separator operating at fixed conditions and defines a pressure boundary condition for the coupled system. The superscripts υ and α denote the respective Newton iteration levels for the reservoir and facility domains. Assuming Neumann boundary conditions at the domain interface, the completed well blocks are treated as source nodes in the facility problem and pressures are the unknown.

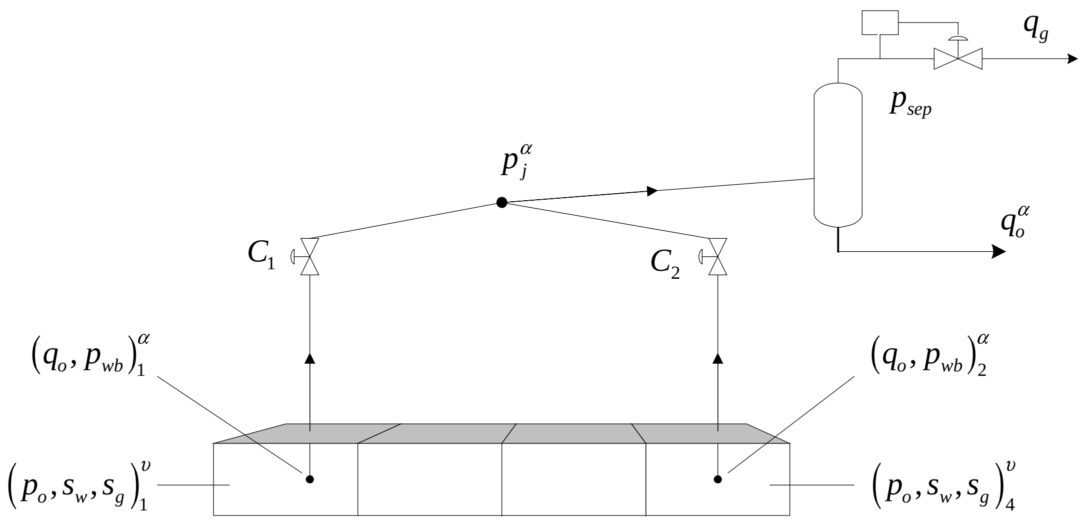


Figure 5.1: Two-well Production System – Explicit Formulation

The global solution procedure for a timestep is summarized as follows:

1. Given well block pressure and saturations at iteration $\upsilon = 0$ and wellbore pressure estimates $\tilde{p}_{wb,1}$ and $\tilde{p}_{wb,2}$ at iteration $\alpha = 0$, compute the well rates $q_{o,1}^\alpha$ and $q_{o,2}^\alpha$ using the well equation (Eq. 3.43).

2. Formulate the facility problem and at each well (source) node *i*, define the boundary condition and unknown as $q_{o,i}^{\alpha}$ and $p_{wb,i}^{\alpha}$, respectively.
3. Take one Newton iteration for the facility problem $(\alpha = \alpha + 1)$ to obtain updated solution estimates $p_{wb,1}^{\alpha}$, $p_{wb,2}^{\alpha}$, p_{j}^{α}, and q_{o}^{α}.
4. If convergence is achieved in the Newton iteration and mass conservation is satisfied in the facility domain, then for the reservoir domain define the boundary condition at each well *i* as either $p_{wb,i}^{\alpha}$ or $q_{o,i}^{\alpha}$, and take Newton steps $(\upsilon = 0,1,...)$ for the reservoir problem until convergence. Note that convergence at a domain interface is defined by $\left|\tilde{p}_{wb,i} - p_{wb,i}^{\alpha}\right| < ptol$, where *ptol* is a specified pressure tolerance.
5. If convergence is not achieved set $\tilde{p}_{wb,i} = p_{wb,i}^{\alpha}$ and using Eq. 3.43, compute new well boundary conditions $q_{o,i}^{\alpha}$, for the facility problem. Go to step 2.

Convergence at the domain interfaces represents a balance between the outflow from the reservoir and inflow to the surface facility. This is commonly shown as the intersection of the inflow performance and tubing intake curves (Brown and Lea, 1985). An analogous procedure can be applied to an explicit coupling where Dirichlet boundary conditions are specified at the domain interface.

The primary motivation for employing an explicit coupling is computational efficiency when compared to the fully implicit formulation. However, in explicit coupling the well boundary conditions for the reservoir domain are based on well block conditions at the beginning of the timestep $(\upsilon = 0)$ and may lead to large material balance errors. As the solution is advanced in the reservoir problem the well block conditions can change, resulting in an imbalance between reservoir outflow and tubing inflow. Examples that demonstrate the inconsistency between the facility and reservoir solutions are presented in Section 5.6.

5.2 Standard Implicit Facility Coupling

Fully implicit coupling allows for placement of the boundary conditions in the mathematical model, which are consistent with the operational constraints and increase the implicitness of the formulation at the wellbore.

The fully implicit coupling implemented for this research is based on a matrix level formulation of the reservoir and facility equations. The linearized reservoir and facility models described in Sections 3.3.1 and 4.4.1, respectively, are employed to formulate the global Newton system. The connectivity between the reservoir and surface facilities is through a conservation relationship similar to Eq. 3.51, and is expressed in residual form as

$$r_{well,c} = \sum_{l} WI_l \sum_{p=1}^{n_p} \beta_p \tilde{T}_{cp,l} \left(p_p - p_{wb} \right)_l - f_t \left(p_{wb,ref}, p_j \right) \tag{5.1}$$

where the first and second terms on right hand side represent the reservoir to well and well to tubing component mass flow rates, respectively. The function f_t represents the pipeflow relationships (Section 4.1.1) used to compute the tubing mass flow rate. In the pipeflow relationship $p_{wb,ref}$ and p_j are the respective upstream and downstream tubing pressures. The downstream tubing pressure typically corresponds to the wellhead or collection point pressure.

Recall that in the well equations $p_{wb,l}$ represents the wellbore pressure in a completed layer *l*. In the well model used for this research, $p_{wb,l}$ is expressed in terms of a pressure at a reference location. The reference pressure $p_{wb,ref}$ represents a primary variable common to the well equation and pipeflow relationship and therefore only one equation is employed for reservoir facility coupling. For production wells the mass balance is applied to the oil component, however as discussed in Chapter 4, the gas component is accounted for throughout the solution procedure. Gas injection wells are only considered in the facility injection network, therefore the mass balance is applied to the gas component.

The derivatives that define the coupling between the well and facility models must also be included in the Jacobian. Assuming only one well block, with primary variables $\left(p, s_g, s_w\right)$, the linearized form of Eq. 5.1 is given by

$$r_{well}^{\upsilon+1} = r_{well}^{\upsilon} + \frac{\partial r_{well}}{\partial p}\Delta p + \frac{\partial r_{well}}{\partial s_g}\Delta s_g + \frac{\partial r_{well}}{\partial s_w}\Delta s_w + \frac{\partial r_{well}}{\partial p_{wb}}\Delta p_{wb} + \frac{\partial r_{well}}{\partial p_j}\Delta p_j \tag{5.2}$$

so that the Newton system is defined as

$$\begin{bmatrix} A_r & A_{rw} & \\ A_{wr} & A_w & A_{wf} \\ & A_{fw} & A_f \end{bmatrix} \begin{bmatrix} \Delta\vec{X}_r \\ \Delta\vec{X}_w \\ \Delta\vec{X}_f \end{bmatrix} = -\begin{bmatrix} \vec{R}_r \\ \vec{R}_w \\ \vec{R}_f \end{bmatrix} \tag{5.3}$$

where $\vec{X}_r = \left\lfloor p_o, s_w, s_g / p_{bp} \right\rfloor_i$, $i = 1,\ldots,n_{xyz}$ are the reservoir variables, and $\vec{X}_w = \left[p_{wf,j} \right]$, $j = 1,\ldots,n_{well}$ and $\vec{X}_f = \left[p_k / q_{p,k} \right]$, $k = 1,\ldots,n_f$ are the well and facility variables, respectively. The submatrices A_{rw} and A_{wf} represent the reservoir to well and well to facility coupling, respectively. The system can be expressed compactly as

$$A_g \Delta\vec{X}_g^{\upsilon} = -\vec{R}_g^{\upsilon} \tag{5.4}$$

where A_g is the Jacobian matrix, $\vec{R}_g$ is the residual vector, and $\Delta\vec{X}_g = \left[\Delta\vec{X}_r, \Delta\vec{X}_w, \Delta\vec{X}_f \right]$. The solution of the combined system constitutes a global update of the Newton step and incorporates the reservoir performance dependence on surface facility conditions through the well equation.

The standard implicit formulation eliminates any errors associated with explicit couplings up to a specified tolerance, however computational efficiency may be affected. The coupled system involves flow equations whose velocity field is described by different relationships. In the reservoir, application of Darcy's law yields a linear relationship between pressure drop and velocity. As discussed in Section 4.1.3, the pressure drop in a pipe is proportional to velocity squared. The Newton step in the reservoir regions can accommodate much larger pressure changes than in the pipe or tubing and therefore the number of Newton iterations required for convergence in each system is different. The surface facility test models presented in Section 4.5 required 4 to 15 Newton iterations for convergence. In contrast, a timestep in the reservoir domain may require only 5 Newton iterations and fully implicit coupling of the reservoir and facility domains can lead to substantial increase in computation time since every iteration involves a linear solve on the global problem domain.

5.3 Adaptive Explicit Facility Coupling

A new formulation that combines the speed of the explicit method with the accuracy of the standard implicit method is presented in this section. The explicit formulation presented in Section 5.1 is attractive due to the computational efficiency obtained by solving the nonlinear reservoir problem independently of the facility problem. However the test problems that are presented in Section 5.6 show considerable material balance errors associated with this formulation. The primary errors are due to significant changes in near wellbore reservoir conditions and for most problems, this can be correlated to the phase behavior and leads to another method for solving coupled systems.

The method consists of applying the standard implicit formulation on the first timestep of the simulation run. Subsequent iterations employ the explicit formulation until changes in wellbore phase behavior are detected, and then the formulation switches to implicit. Once the system has stabilized the formulation returns to the explicit method.

The criteria for switching from an explicit to implicit formulation is based upon the solution gas-oil ratio, $R_{go,i}$, for any production well *i*. For $R_{go,i}^{n-1} \neq 0$, a parameter ς_i^n defined by

$$\varsigma_i^n = \frac{R_{go,i}^n - R_{go,i}^{n-1}}{R_{go,i}^{n-1}} \cdot 100 \quad , i = 1, \ldots, n_{well} \tag{5.5}$$

is compared with a user supplied percent change in solution gas-oil ratio, $\varsigma_{\max}$ and if for any well *i*, $\varsigma_i^n > \varsigma_{\max}$, the formulation is set to implicit.

Similarly, to switch back to the explicit formulation the following condition must be satisfied

$$\frac{\varsigma_i^n - \varsigma_i^{n-1}}{\varsigma_i^{n-1}} \cdot 100 < 2\varsigma_{\max}\ , \quad \varsigma^{n-1} \neq 0,\ i = 1,\ldots,n_{well} \tag{5.6}$$

The threshold $2\varsigma_{\max}$ is selected for convenience and the factor is included to prevent premature activation of the explicit formulation.

The performance of the method is related directly to the number of standard implicit iterations. Since each local well region can develop two-phase flow at different times, the method may require a significant number of standard implicit iterations and the performance will approach that of the standard implicit formulation. The method presented in the next section is designed to accelerate the standard implicit iterations.

5.4 Preconditioning Method

A new method is presented for solving the fully coupled reservoir and surface facility problem based on preconditioning the standard implicit iterations of the fully coupled and adaptive explicit facility formulation described in the previous sections. The preconditioner is constructed by decomposing the global reservoir domain into local reservoir domains (subdomains), each containing a well. The local domains are of the same grid resolution as the global reservoir domain. The well and facilities domains are solved implicitly as a locally fully coupled system which establishes a preconditioned initial solution for a standard or adaptive implicit treatment of the reservoir flow field. The local subdomain boundary conditions are established from fine grid information or by a coarse grid solve. These components are used to accelerate the Newton step locally in the subdomain and facilities. The method employs several numerical techniques which when combined provide a broad range of options for improving performance.

The preconditioning method is developed in the context of the standard implicit formulation for the reservoir flow field. Results based on the application of the method are presented in Section 5.6.

5.4.1 Subdomain Boundary Conditions

The subdomain component in the preconditioner is designed to provide local reservoir information for the solution of the facility model. Each subdomain requires specification of reservoir boundary conditions that correspond to either the external reservoir boundary or to an internal subdomain interface. To initiate the solution process for the first timestep a coarsened problem is solved over one timestep to obtain estimates of the fine grid pressure and saturation distribution that can be used to construct Dirichlet or Neumann boundary conditions at the subdomain interfaces. After the first timestep the subdomain boundary conditions, of either Dirichlet or Neumann type are constructed from the reservoir fine grid properties. Each type of boundary condition is further discussed in this section and comparison test results are presented in Section 5.5.1.

Coarse Grid Generated Boundary Conditions for the Well Subdomains

The coarse grid solution algorithm is based on an algebraic coarsening technique applied to the global fine grid Jacobian matrix. The coarsening is restricted to the areal dimensions of the grid and only one well per coarse cell is allowed. As shown in Fig. 5.2 the coarse grid problem is fully coupled to the facility model.

The coarse grid operator is derived from multigrid principles and leads to a Galerkin matrix (Briggs, 1987) which is defined by

$$A^{h'} = I_h^{h'} A_g I_{h'}^h \tag{5.7}$$

$$I_{h'}^h = \left(I_h^{h'}\right)^T \tag{5.8}$$

where h and h' are the respective number of unknowns on fine and coarse grids. $I_h^{h'}$ is a linear restriction operator from the fine to the coarse grid and $I_{h'}^h$ is a piecewise constant prolongation operation from the coarse to the fine grid. The linear restriction operator sums all the fine grid cell Jacobian coefficients that define a coarse grid cell to obtain a coarse grid cell coefficient. Similarly the coarse to coarse grid cell dependency

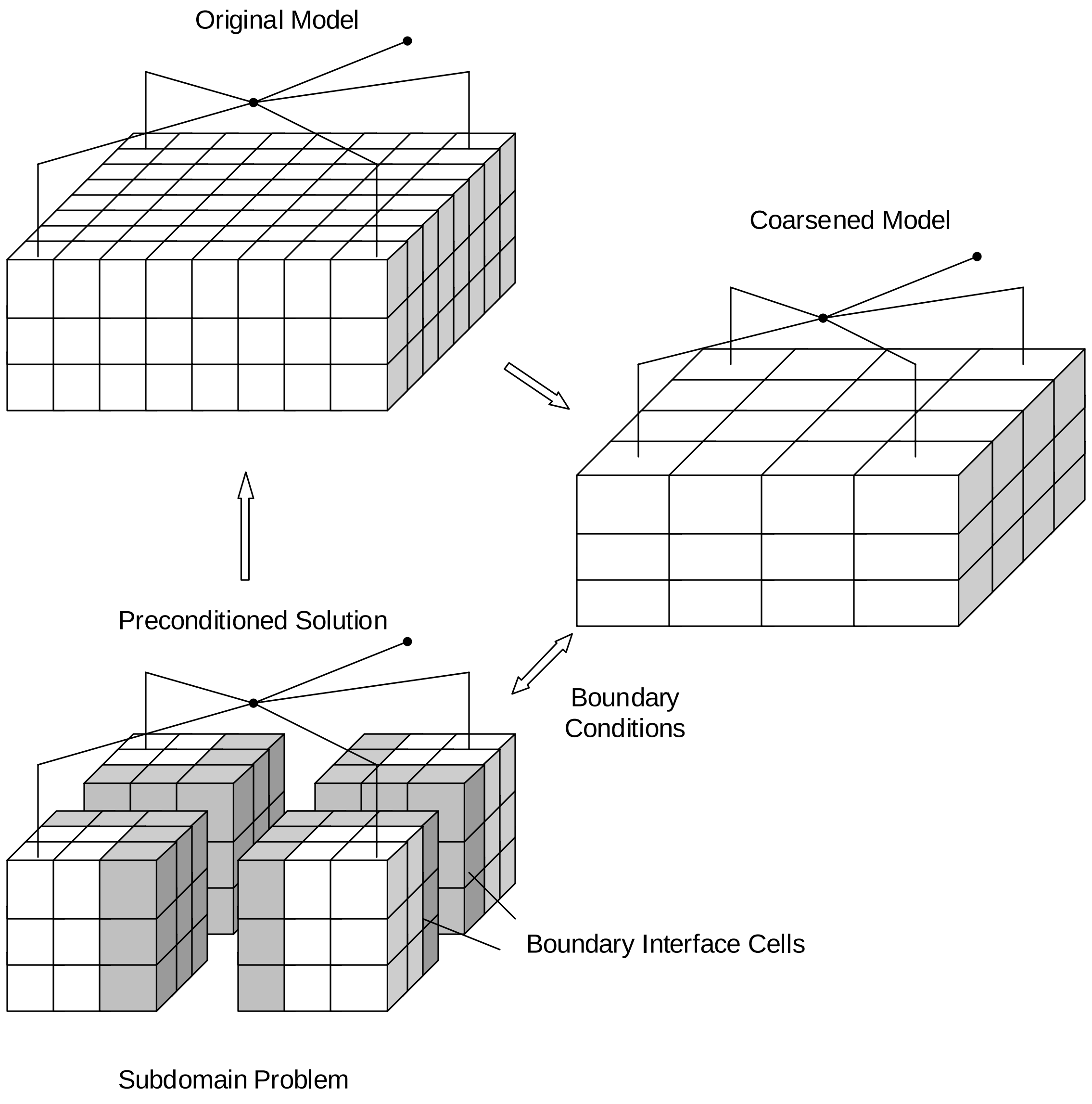


Figure 5.2: Preconditioning Method (Conceptual Diagram)

coefficients are obtained by summing the respective fine to fine grid cell dependency coefficients. The prolongation operation assigns the solution for a coarsened cell to all the fine grids cells that define the coarsened cell. Modifications to the operators are required when a coarse cell contains fine cells with different phases. For example, if a fine cell contains free gas the primary variable is gas saturation, otherwise it is bubble point pressure. Threshold criteria are used to ensure consistency of the primary variables and thus Jacobian terms for the coarse grid cell. A detailed discussion of restriction and prolongation operators is provided by Briggs (1987).

A coarse grid residual vector is similarly constructed using the restriction operator. The coarse grid solution can then be obtained by solving

$$X^{h'} = \left(I_h^{h'} A_g I_{h'}^{h} \right)^{-1} \left(I_h^{h'} R_g \right) \tag{5.9}$$

and the interpolated fine grid solution is

$$X_g = I_{h'}^{h} X^{h'} \tag{5.10}$$

The fine grid solution is applied to the subdomain boundary cells, which are then used to compute Dirichlet or Neumann boundary conditions.

Fine Grid Generated Boundary Conditions for the Well Subdomains

After the first timestep of the simulation subdomain boundary conditions generated using boundary cell information generated from the coarse grid problem introduce smoothed values at the interfaces. An alternative is to construct boundary conditions based on the previous iteration fine grid boundary values, which contains the best information available. A test model presented at the end of this section provides a comparison of the approaches and establishes the configuration that will be used in all subsequent test results.

Subdomain Solve

Local fine grid subdomains are defined for each well. The wells and thus the subdomains are coupled via the facilities. The respective local fine grid solutions are calculated with boundary conditions supplied by the initial coarse grid solution and facility constraints. This collective subproblem provides a good starting solution for the fully coupled reservoir and facility problem. Different variants of the strategy can be devised.

The subproblem unknowns can be expressed as

$$\vec{X}_{sp} = \left[\vec{X}_{sd,i}, \vec{X}_{f} \right],\ i = 1,\ldots,n_{sd} \tag{5.11}$$

where n_{sd} is the total number of locally refined reservoir subdomains and $\vec{X}_{sd,i}$ is the vector of unknowns for the local well subdomains. The Newton system for the linearized volume balance equations may be expressed in matrix form as

$$J(\vec{X}_{sp}^{(\alpha)})\delta\vec{X}_{sp} = -R(\vec{X}_{sp}^{(\alpha)}),\ \alpha = 0,1,\ldots \tag{5.12}$$

where $\delta\vec{X} = \vec{X}_{sp}^{(\alpha+1)} - \vec{X}_{sp}^{(\alpha)}$.

The subproblem solution is used to precondition the global solution for the fully coupled solve. The procedure is described by the flow chart in Figure 5.3. Guidelines for defining preconditioning frequency and subdomain boundary condition type are presented in Section 5.5.

5.4.2 Preconditioned Adaptive Implicit Flow Modeling

The preconditioning technique developed for accelerating the standard implicit facility coupling formulation is a natural candidate for use in an adaptive implicit formulation. The convergent flow near the wellbore regions will result in high velocity regions that require an implicit treatment. The initial boundary for implicitness is chosen to coincide with the well subdomains defined by the preconditioner, resulting in a preconditioning of the fully coupled standard adaptive implicit problem. Results based on this technique are presented by Byer *et al*. (1999) and are summarized here.

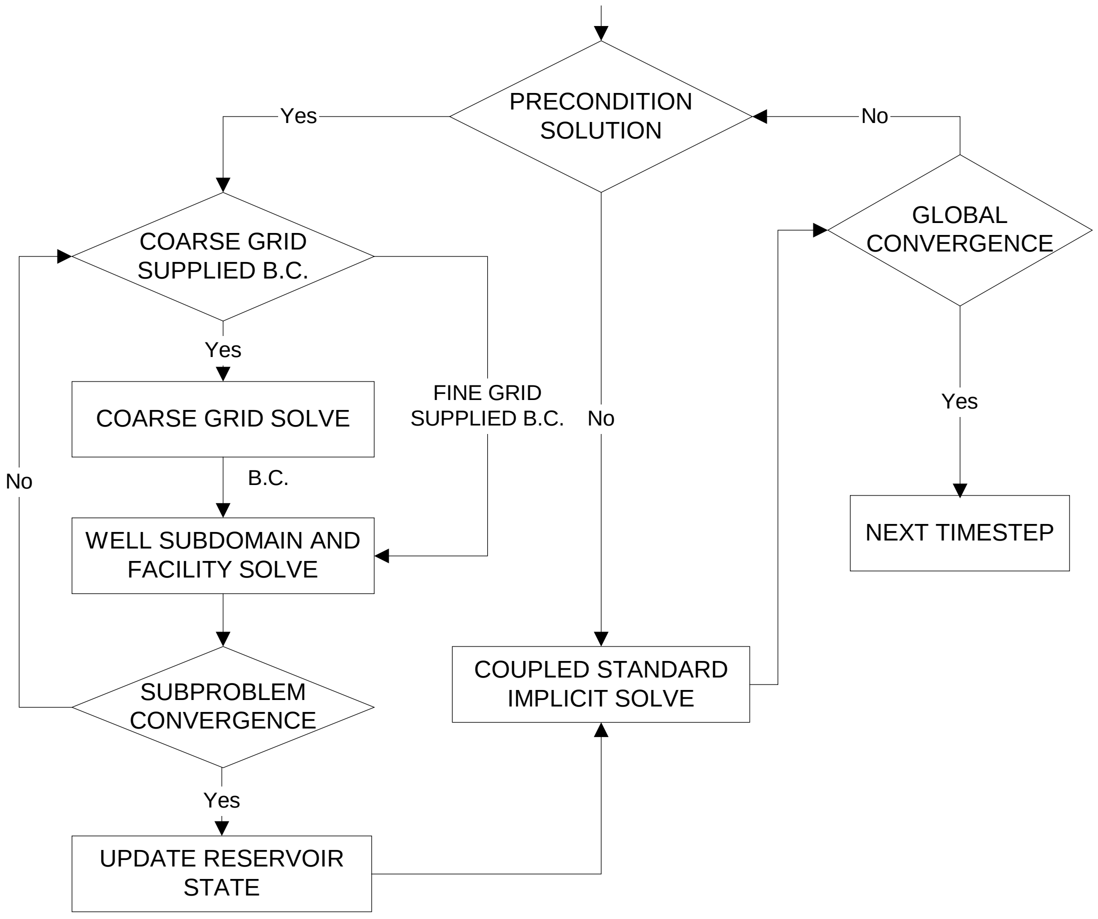


Figure 5.3: Flowchart of Preconditioned Newton Method

A combined reservoir and facility model problem that consists of ten wells tied into a gathering system was developed to demonstrate the performance characteristics of the preconditioning method when applied to standard and adaptive implicit formulations. A fixed rate boundary condition (16000 stb/day) was specified at the sink node in the gathering system. Ten injection wells are included to provide pressure support, but are not included in the facilities configuration.

The first test used the model problem to compare an explicit coupling versus the standard and preconditioned standard implicit formulations. The explicit facility coupling yielded a maximum error of 8% in oil production rate and 6% in pressure at the facility boundary condition. The fully implicit formulation did not introduce any additional error relative to the convergence tolerances. However there was a significant increase in CPU time. By comparison, the preconditioned standard implicit formulation reduced the CPU requirements to that of explicit coupling.

The second test was designed to demonstrate the effectiveness of the preconditioned adaptive implicit formulation at reducing the computational effort in the presence of convergent flow near the wellbore. The adaptive implicit formulation was effective in maintaining an appropriate level of implicitness around the convergent flow regions. The average percent implicitness ranged from 11% to 21% according to the subdomain size. The optimal preconditioner with respect to total CPU time resulted in a 16% reduction in CPU time compared to the standard adaptive implicit formulation. The relative CPU time for the preconditioned fully implicit cases indicated that the optimal preconditioner reduced the CPU requirements by 28% when compared to the standard fully implicit case. The differences between the preconditioned fully implicit and preconditioned adaptive implicit cases can be understood by considering the ratio of the time required by the preconditioner and linear solve. On average, this value was 18% for fully implicit and 42% for adaptive implicit formulation. The preconditioned method is designed to reduce the impact of the facility model through reduced standard implicit solves. The margin for improvement reduces as the size of the standard adaptive implicit solve is decreased, thus accounting for the differences in relative CPU time, however the preconditioner still provides a beneficial reduction in CPU time.

5.5 Improving Performance

The preconditioning method contains several components for improving performance. The efficiency of the method is determined by the subdomain size, boundary condition type, and preconditioning frequency, and the interactions between these components. Guidelines and criteria for defining these options were examined using the test model shown in Fig 5.4. The facility configuration is defined by a producer and injector pair of wells located in a channeled reservoir. Constant pressure boundary conditions are assigned to both wells and the field is developed for 2000 days. Model data are shown in Tables 5.1-2. Reservoir rock-fluid properties are listed in Appendix B.

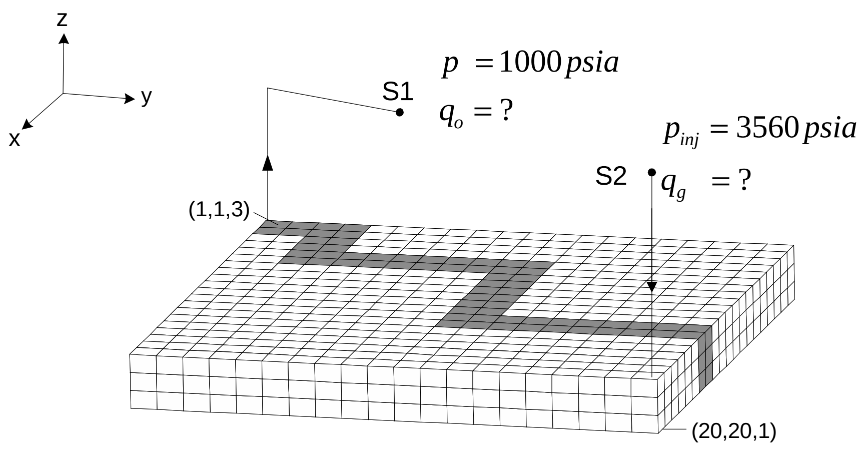


Figure 5.4: Reservoir and Facilities Configuration - Test Model 1

Table 5.1: Reservoir Data – Test Problem 1

Parameter			
nx,ny,nz	20x20x3	depth to top of reservoir	5260'
dx=dy	500'	channel permeability, kx=ky=kz	800 md
dz	30'	permeability outside channel	
porosity @ 14.7 psia	0.20	kx=ky	200 md
pressure @ 8325'	3500 psia	kz, layer 1	30 md
connate water saturation	0.12	layer 2	20 md
initial oil saturation	0.88	layer 3	50 md

Table 5.2: Surface Facilities Data – Test Problem 1

Source/Sink	Coordinate location[1]	Boundary Conditions	
S1	(5000,5000,0)	S1	1000.0 psia
S2	(9750,9750,0)	S2	3560 psia

[1]reference location is the corner of cell (1,1,3) with coordinates (0,0,0), expressed in units of feet.

	Well block	Tubing data[2]		Wellhead to source/sink node[2]	
Well	(i,j,k)	Vertical length	Diameter	Horizontal length	Diameter
1	(1,1,1)	5335'	4.5"	6363.96'	4.5"
2	(20,20,3)	5275'	3.5"	-	-

[2]absolute roughness equals 0.0001'

Subdomain Boundary Conditions

The total Newton iterations required for the preconditioned and standard implicit formulations are compared to measure the effect of subdomain boundary condition type on preconditioner performance. Several test runs are performed that employed a range of subdomain sizes. Dirichlet or Neumann boundary conditions are specified at the subdomain-reservoir interfaces and preconditioning is performed only on the first Newton iteration of each timestep. The results are shown in Fig. 5.5 and for reference, the number of iterations required for a run without preconditioning is shown. In all cases, Dirichlet boundary conditions resulted in better performance than Neumann conditions and reduced the number of standard implicit fully coupled iterations. In particular, when using 2x2x3 subdomain and Neumann conditions the number of iterations exceeded that of the case without preconditioning. In this case the application of the preconditioner with Neumann boundary conditions resulted in a subdomain solution which provided

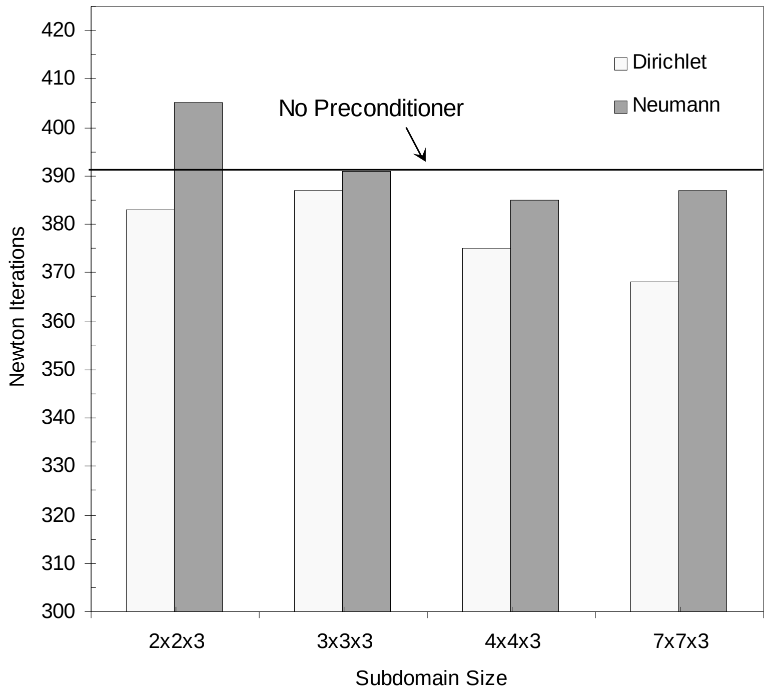


Figure 5.5: Effect of Subdomain Boundary Condition Type on Convergence Rate

minimal acceleration and required considerable correction by the fully coupled step, and on a few iterations required a timestep reduction, which was not present with Dirichlet boundary conditions. The individual Newton step performance did not provide any basis for developing boundary condition selection criteria and therefore all subsequent runs employ Dirichlet boundary conditions.

In each test run the first two Newton iterations of the simulation employed fine grid information obtained from the coarse grid problem to specify Dirichlet or Neumann conditions at the subdomain boundaries. Similar to the previous results, Dirichlet boundary conditions are preferable and the application of the coarse grid solution generally reduces the number of iterations by one to two when compared to boundary conditions constructed with fine grid initial data. Consequently, it is recommended that the coarse grid based boundary conditions be applied at the beginning of the simulation due to the absence of higher quality fine grid information and then on subsequent timesteps previous iteration fine grid information should be employed.

5.5.1 Subdomain Size and Preconditioning Frequency

The effect of subdomain size and preconditioning frequency on the Newton convergence behavior are examined as separate components to isolate their impact on the method. Based on the results of numerous simulations using the model presented at the beginning of Section 5.5, selection criteria for each component are developed and tested.

Subdomain Size

The subdomain component in the preconditioning method is used to capture local reservoir behavior around the wellbore in the solution of the facility problem. The impact of the subdomain size on convergence rate is shown in Fig. 5.6. The first five cases are defined with an equivalent number of cells in the *x* and *y* dimensions and the results show constant or improved Newton convergence rate with increasing subdomain size. As the size of the subdomain increases the implicitness of the preconditioner increases and results in a higher quality preconditioned solution. The cost of the preconditioner in terms of Newton convergence rate increases with subdomain size, which is due to convergence requirements on a larger portion of the reservoir. Also, the CPU requirements for the preconditioner will increase due to the larger Jacobian in the linear solve. The results from the first five test runs indicate that the optimal subdomain size is based on the balance between reduced standard implicit iterations and increased iterations in the preconditioners, however to establish an optimal configuration requires several trial runs and is also problem dependent.

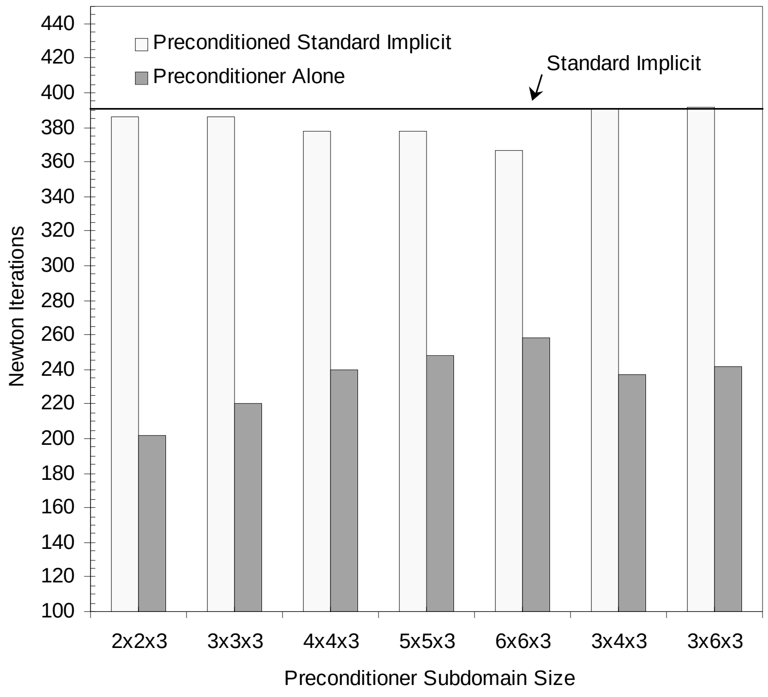


Figure 5.6: Sensitivity of Newton Convergence Rate to Subdomain Size

The last two cases in Fig 5.6 (3x4x3 and 3x6x3) were included to show that placement of the subdomains to coincide with dominant flow paths does not necessarily equate to a higher quality preconditioner. For each of these runs the Newton convergence rate for the standard implicit problem was similar to the 3x3x3 case. However there were a few timesteps in which the preconditioned solution considerably slowed convergence rate. While it is difficult to define the cause precisely, examination of saturation maps indicates that injection gas was moving into the producer region, but just outside the subdomain boundaries. This suggests that the size of the subdomain was not able to incorporate enough of the local phase behavior to obtain a quality preconditioned solution. Thus a larger subdomain size (3x6x3 vs. 3x3x3) does not imply better convergence. This suggests that a criterion for either defining the subdomain size based on local phase behavior, or alternatively, detecting when to avoid preconditioning, should be developed. In this research the later approach is employed since the criterion for preconditioning frequency was determined to be an important component in the efficiency of the preconditioning method. This aspect is discussed further here.

Development of Preconditioning Strategies

In the tests just demonstrated the best preconditioning frequency was selected by experiment. Examination of two test runs, which use the same subdomain size (7x7x3) but different preconditioning frequencies, serve to illustrate the criteria design. The following strategies have been considered:

1. Early Preconditioning - precondition only the first two iterations of each timestep.
2. Delayed Preconditioning - precondition only two iterations of the timestep, but delay preconditioning until after the first two Newton iterations.
3. Variable Preconditioning – precondition based on previous timestep preconditioner performance and the current timestep global convergence behavior.

Strategies 1 and 2 – Early and Delayed Preconditioning

The first two strategies both apply the preconditioner for a maximum of two consecutive iterations per timestep, the first strategy preconditions the first two Newton iterations while the second delays preconditioning until after the first two Newton iterations. The performance of each strategy is evaluated with respect to standard implicit results. The strategies are then contrasted to identify timesteps that show different preconditioning behavior.

Each strategy is compared to the standard implicit results based on the difference in Newton iterations, d_{NI}^{n}, defined as

$$d_{NI}^{n} = \upsilon_{t,PcSI}^{n} - \upsilon_{t,SI}^{n} \tag{5.13}$$

where $\upsilon_{t,PcSI}^{n}$ and $\upsilon_{t,SI}^{n}$ are the respective total number of preconditioned standard implicit (PcSI) and standard implicit (SI) Newton iterations required for timestep *n*. The values for d_{NI}^{n} are interpreted as follows:

1. If $d_{NI}^{n} > 0$ then the standard implicit formulation performed better than the preconditioned formulation and required d_{NI}^{n} less Newton iterations for timestep *n*.
2. If $d_{NI}^{n} = 0$ then preconditioning had no benefit and both formulations required the same number of Newton iterations for timestep *n*.
3. If $d_{NI}^{n} < 0$ then the preconditioned formulation performed better than the standard implicit formulation and required d_{NI}^{n} less Newton iterations for timestep *n*.

The differences in global Newton iterations in the preconditioned and nonpreconditioned runs for each strategy are shown in Figs. 5.7 and 5.8. Preconditioning the first two Newton iterations resulted in slower convergence rates than without preconditioning throughout most of the simulation (Fig 5.7). Delayed preconditioning indicates much better performance, however occasionally preconditioning slowed the rate of convergence (Fig 5.8). Examination of these timesteps shows that after returning from the preconditioner, a large correction from the standard implicit solve was required, indicating that even after two standard implicit iterations, reservoir conditions were not stabilized enough to obtain a good preconditioned solution.

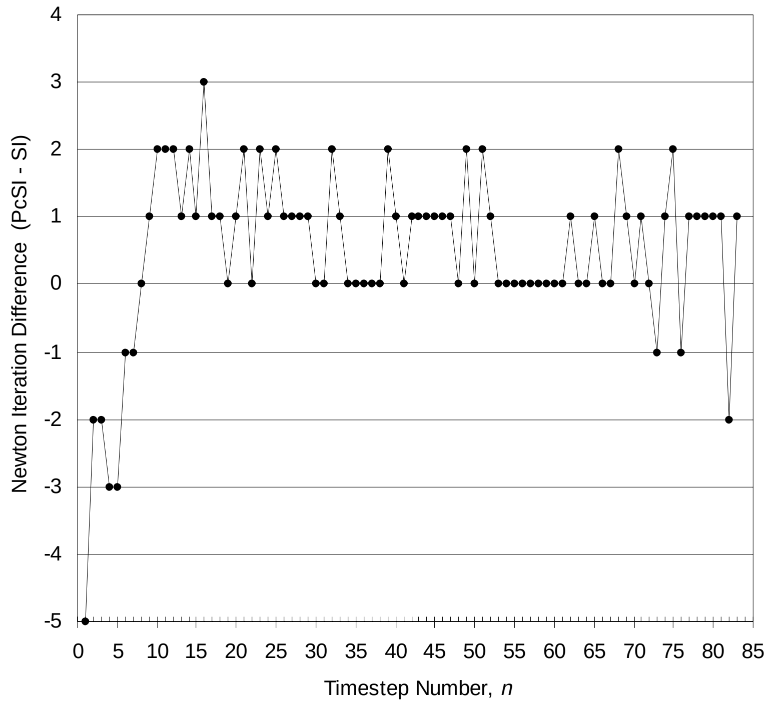


Figure 5.7: Newton Iteration Difference, $d_{NI} = \upsilon_{t,PcSI}^{n} - \upsilon_{t,SI}^{n}$, ($\upsilon_{t,PcSI}^{n}$, Early PcSI)

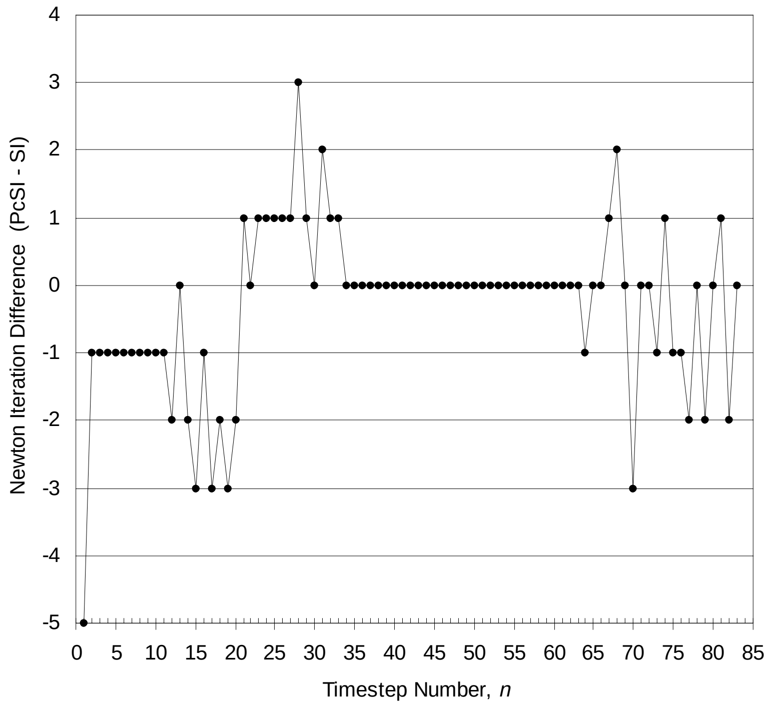


Figure 5.8: Newton Iteration Difference, $d_{NI} = \upsilon_{t,PcSI}^{n} - \upsilon_{t,SI}^{n}$, ($\upsilon_{t,PcSI}^{n}$, Delayed PcSI)

Similar to the previous comparisons, the early and delayed strategies can be contrasted by examining the difference between the number of Newton iterations required by each strategy. In this comparison the difference in Newton iterations, d_{NI}^{n}, is defined as

$$d_{NI}^{n} = \upsilon_{t,PcSI(Early)}^{n} - \upsilon_{t,PcSI(Delayed)}^{n} \tag{5.14}$$

where $\upsilon_{t,PcSI(Early)}^{n}$ and $\upsilon_{t,SI(Delayed)}^{n}$ are the total number of preconditioned standard implicit Newton iterations required for timestep *n* as a result of the respective early and delayed strategies. The values for d_{NI}^{n} are then interpreted as follows:

1. If $d_{NI}^{n} > 0$ then the delayed preconditioning strategy performed better than the early strategy and required d_{NI}^{n} less Newton iterations for timestep *n*.
2. If $d_{NI}^{n} = 0$ then the early and delayed preconditioning strategies resulted in the same performance.
3. If $d_{NI}^{n} < 0$ then the early preconditioning strategy performed better than the delayed strategy and required d_{NI}^{n} less Newton iterations for timestep *n*.

The differences are shown in Fig. 5.9. The material balance error and linear solution data for both runs were examined to seek correlations with the preconditioning quality. The primary indicator for whether to apply preconditioning on the first Newton step was found to be the material balance error. A large material balance error is equivalent to a large residual vector for the linear solve and can result in a significant change in the primary variables. In this case the local subdomain information can be of poor enough quality to result in a bad preconditioned solution and can even lead to slower convergence when compared to no preconditioning. However, even if the initial material balance error is within a specified limit, as the Newton iteration progresses reservoir conditions can change and adversely affect preconditioning.

Both strategies contained iterations where preconditioning increased the maximum primary variable change or material balance error, which resulted in an increased number of Newton iterations. This condition becomes an integral component of the variable preconditioning strategy discussed next.

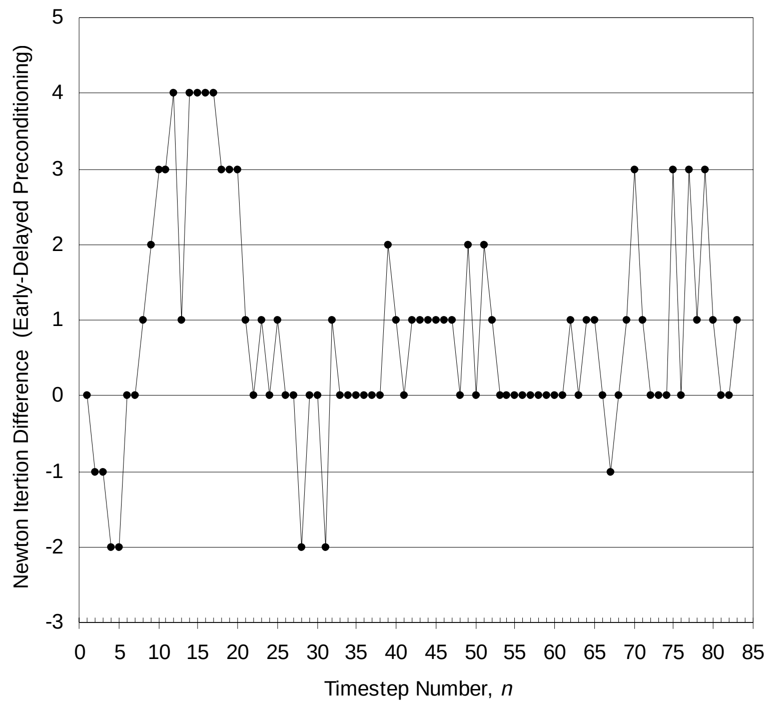


Figure 5.9: Newton Iteration Difference – Early Minus Delayed Preconditioning

Strategy 3 – Variable Preconditioning

The variable preconditioning strategy developed for this research employs multiple criteria applied over a timestep to determine if preconditioning should be applied. This strategy is employed in the test problems in Section 5.6. A detailed presentation and flowchart of the strategy is contained in Appendix C and summarized here.

The strategy contains criteria that can be categorized by the use of either previous or current timestep information. The criteria based on current timestep information are used to initiate or delay preconditioning within the current timestep. The criteria are based on the premise that application of the preconditioner should not increase material balance errors or delay Newton convergence rate in the reservoir domain.

Performance data related to the preconditioning strategy at earlier timesteps is used to help define the first iteration at which preconditioning can be considered for the current timestep. The total number of Newton iterations and sequence of preconditioned iterations from the previous timestep are examined. If the total number of Newton iterations is less than a threshold value then the previous timestep's strategy is considered a success for that threshold value and criteria designed to maintain or improve the converge rate are employed. Otherwise the sequence of preconditioned Newton iterations is examined to determine whether preconditioning should be applied earlier or later in the current timestep. Additional criteria based on the previous two or three timesteps convergence behavior are used to prevent stagnating or oscillating behavior in the Newton convergence rate.

The results of applying the variable preconditioning strategy to the above test model are compared against the strategy which employed delayed preconditioning. The variable strategy reduced the standard implicit iterations (Fig. 5.10) and the preconditioner iterations (Fig. 5.11) considerably for all the cases. Also recall that poor performance of the 3x4x3 and 3x6x3 cases shown in Fig. 5.6 was due to preconditioning that slowed convergence. The new criteria were able to apply the preconditioning selectively and the convergence rate of these cases is improved greatly and demonstrates the ability of the method to accelerate convergence for relatively simple coupled models. Tests of the method with more complex models are described in Section 5.6.

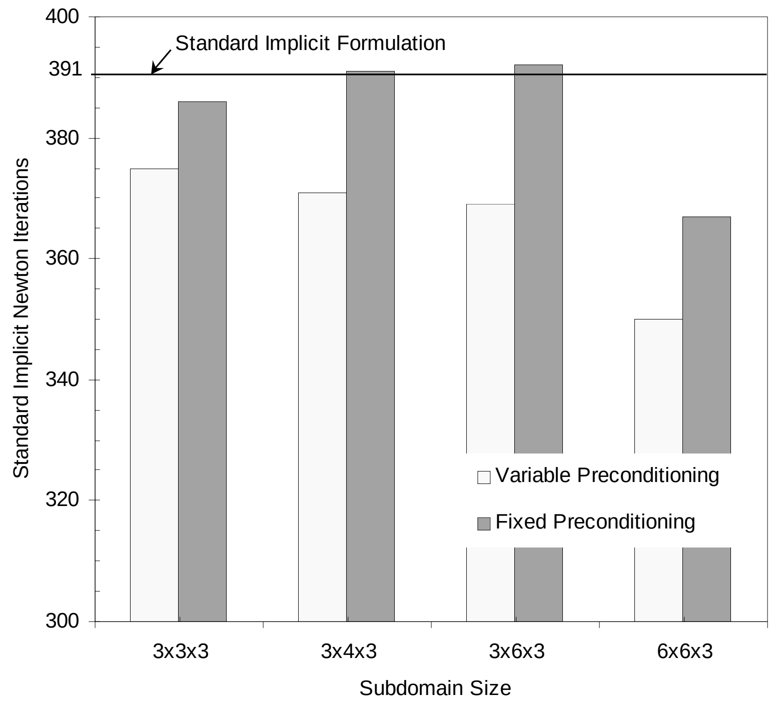


Figure 5.10: Standard Implicit Iterations Fixed vs. Variable Preconditioning

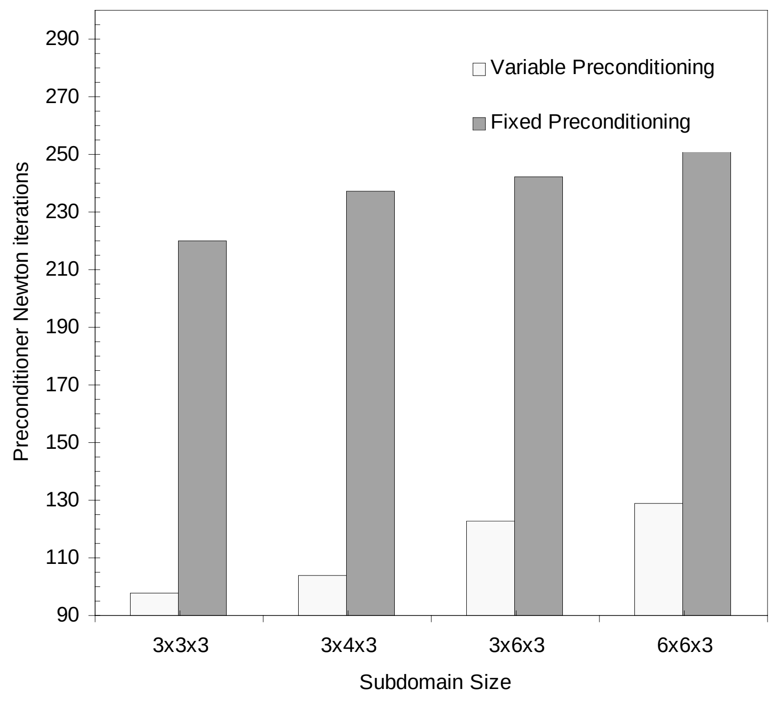


Figure 5.11: Preconditioned Iterations Fixed vs. Variable Preconditioning

5.6 Comparison of Coupled Formulations

The formulations described earlier in this chapter were applied to two coupled reservoir and surface facility test models. The test problems are presented separately, with a description of the model and a brief discussion of operational behavior preceding the discussion and comparison of the solution methods.

The accuracy of the explicit and adaptive explicit facilities methods are compared against the standard implicit formulation. The comparisons are based on results obtained from the nonpreconditioned runs, which were identical to those of the preconditioned runs.

The computational efficiency of the formulations is compared with respect to Newton convergence rate and CPU time. The results for the nonpreconditioned explicit, standard implicit and adaptive explicit formulations are presented first. Next, results from applying the new preconditioning method to the standard implicit and adaptive explicit facility formulations are compared against the respective nonpreconditioned formulations. Additionally, the performance of the preconditioning is tested with respect to variations in well subdomain size.

The analysis of the formulations includes discussion that refers to individual formulation components. To facilitate discussion, component definitions and membership for each formulation are summarized in Table 5.3.

5.6.1 Problem 1 - Model Description

The first test problem was designed to simulate production from three well groups over a ten year horizon. An additional injection group is included to provide pressure maintenance. The reservoir is characterized by an isotropic layered permeability field with homogenous vertical permeability. The rock-fluid and PVT properties are listed in Appendix B. The reservoir and surface facilities layout and data are shown in Fig. 5.12 and Tables 5.4-6.

Table 5.3: Major Formulation Components and Definitions

	Component	Fully Coupled	Preconditioner	Reservoir Domain	Facilities Domain
Coupling Formulation					
Explicit Facilities	(Exp-Fac)			x	x
Standard Implicit Facilities	(SI-Fac)	x			
Adaptive Explicit Facilities	(AE-Fac)	x		x	x
Preconditioned Standard Implicit Facilities	(PcSI-Fac)	x	x		
Preconditioned Adaptive Explicit Facilities	(PcAE-Fac)	x	x	x	x

Component	Definition
Fully Coupled	global reservoir domain with fully implicit facilities coupling
Preconditioner	well subdomains with fully implicit facilities coupling
Reservoir Domain	global reservoir domain with well boundary conditions
Facilities Domain	facilities with source/sink boundary conditions corresponding to production/injection wells

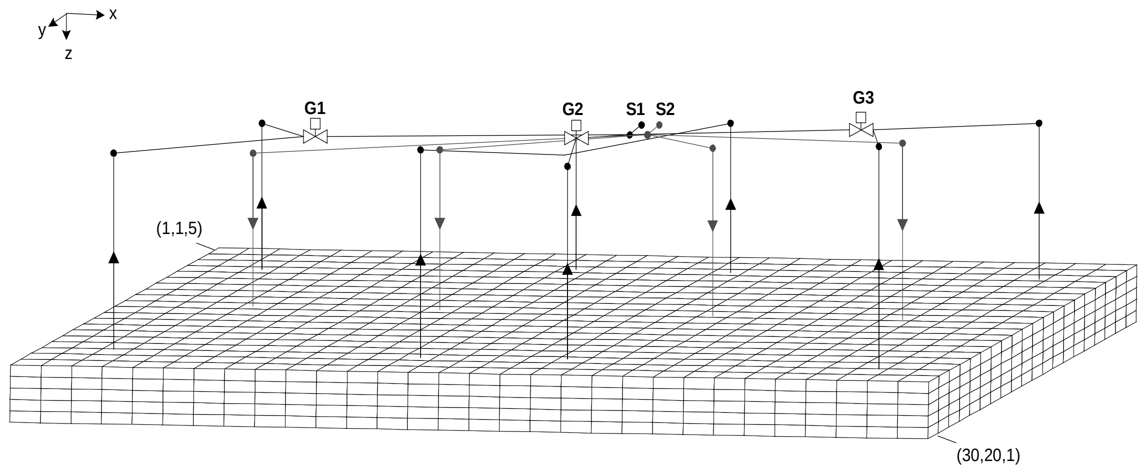


Figure 5.12: Reservoir and Facilities Model - Problem 1

Table 5.4: Reservoir Data –Problem 1

Parameter			
nx,ny,nz	30x20x5	depth to top of reservoir	5260’
dx=dy	500’	kx=ky, layer 5	600 md
dz	18’	layer 4	500 md
porosity @ 14.7 psia	0.30	layer 3	300 md
Pressure @ 8325’	3500 psia	layer 2	200 md
Initial water saturation	0.12	layer 1	300 md
initial oil saturation	0.88	kz	50 md

Table 5.5: Surface Facilities Data –Problem 1

Junction/Source/Sink	Coordinate location[1]		
G1	(2500,5000,0)	(not shown in Fig.) G5	(7500,2000,0)
G2	(7500,5000,0)	S1	(7500,1000,0)
G3	(12500,5000,0)	S2	(7500,1000,0)
(not shown in Fig.) G4	(7500,2000,0)		

[1]reference location is the corner of cell 1,1,1 with coordinates (0,0,0), expressed in units of feet.

Surface pipe data[2]		Boundary Conditions	
Production Gathering Network	Diameter		
G1 to G5	6.5”	S1	1000.0 psia
G2 to G5	6.5”		
G3 to G5	6.5”		
G5 to S1	7.5”		
Injection Network			
S2 to G5	8.5”	S2	3505.0 psia

[2]absolute roughness of all pipe is 0.0001’

Operational data		Choke data	
Production group	Minimum rate (stb/day)	Initial choke setting	Discharge coef.
G1	10000.0	0.65”	0.85
G2	15000.0	0.65”	0.85
G3	10000.0	0.65”	0.85

Table 5.6: Surface Facilities Data –Problem 1

		Well block	Tubing data[2]		Wellhead to junction node[2]	
Group	Well	(i,j,k)	Vertical length	Diameter	Horizontal length	Diameter
G1	1	(3,3,1)	5341’	5.5”	3640’	6.5”
	2	(3,18,1)	5341’	5.5”	4123’	6.5”
G2	1	(13,3,1)	5341’	5.5”	3640’	6.5”
	2	(18,3,1)	5341’	5.5”	3807’	6.5”
	3	(13,18,1)	5341’	5.5”	4123’	6.5”
	4	(18,18,1)	5341’	5.5”	4272’	6.5”
G3	1	(28,3,1)	5341’	5.5”	3807’	6.5”
	2	(28,18,1)	5341’	5.5”	4272’	6.5”
G4	1	(5,10,5)	5269’	4.5”	5830’	4.5”
	2	(11,10,5)	5269’	4.5”	3605’	4.5”
	3	(20,10,5)	5269’	4.5”	3905’	4.5”
	4	(26,10,5)	5269’	4.5”	6426’	4.5”

The operational objective is to produce at least 35,000 stb/day from the reservoir with the facility constraint defined by constant separator pressure. The total production rate is distributed among three well groups and the minimum target rate for each group is controlled using a variable choke. Initially all the chokes are set at 50% open and if the target specified rate for that group is not achieved the choke is adjusted to meet the target rate. Fig. 5.13 shows the adjustments made on each choke to maintain the group production targets shown in Fig 5.14. At the beginning of production the chokes are adjusted continuously until the injection program reverses the pressure decline. Once the production meets or exceeds the minimum target, no choke adjustments are required until a free gas phase begins to flow from the reservoir. The chokes are eventually open 100% and the minimum target cannot be maintained. The symmetry of the well placement accounts for the identical choke setting and rate profiles for groups G1 and G2.

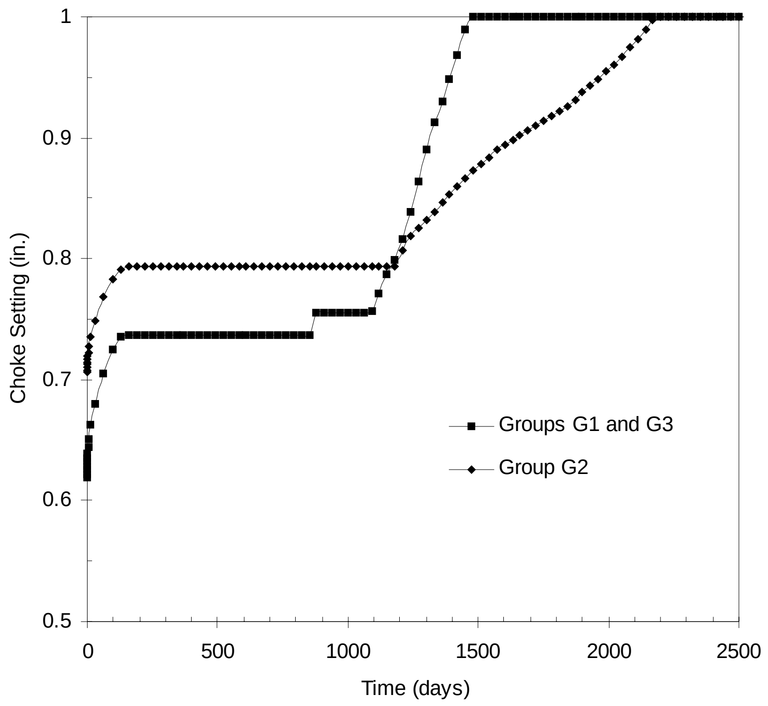


Figure 5.13: Variable Choke Performance – Problem 1

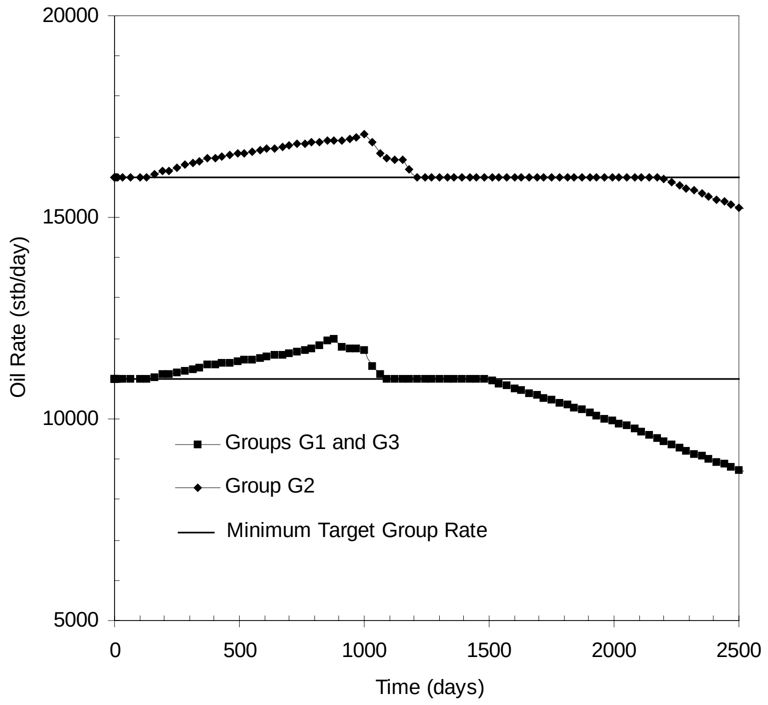


Figure 5.14: Group Oil Rate Production Profiles – Problem 1

5.6.2 Problem 1 - Accuracy of Formulations

Explicit Facilities

Recall that the explicit coupling consists of solving the surface facilities problem where previous timestep reservoir conditions are used to determine flow into the wellbore. The results of the facility solve establish Neumann or Dirichlet boundary conditions for each well. In this problem, Dirichlet conditions are employed since constant pressure boundary conditions are specified for both the sink (*S1*) and source nodes (*S2*). The well flowing pressure is fixed throughout the timestep. There is an error in the flow rate when comparing, 1) the beginning of timestep rate computed from the explicit facilities solve and, 2) the rate obtained at the end of the timestep from the reservoir solve. This error demonstrates the inconsistency that exists within the explicit formulation and an additional comparison against the rate obtained from standard implicit formulation is required to establish the error with respect to formulations. Since the magnitude and profile of these errors (beginning of timestep explicit facility rate vs. end of timestep reservoir rate, and beginning of timestep explicit facility rate vs. standard implicit rate) is very similar, only comparisons based on the later combination are presented (Fig. 5.15). The well production rates are identical for several of the wells and only a subset of the errors is shown. During the time period between 800 and 1300 days the oil rate errors ranged from 5 to 11%. A similar error profile is shown in the results presented by Schiozer (1994). The time period corresponds to a transition from single-phase oil to two-phase oil-gas flow in the wellbore.

Adaptive Explicit Facilities

An adaptive explicit facilities formulation is introduced that is an ideal candidate for this model since the rate error is small outside of the transition period. The adaptive method will apply an explicit facilities formulation outside of the transition period and a standard implicit formulation within the transition period. In this problem, the method switches

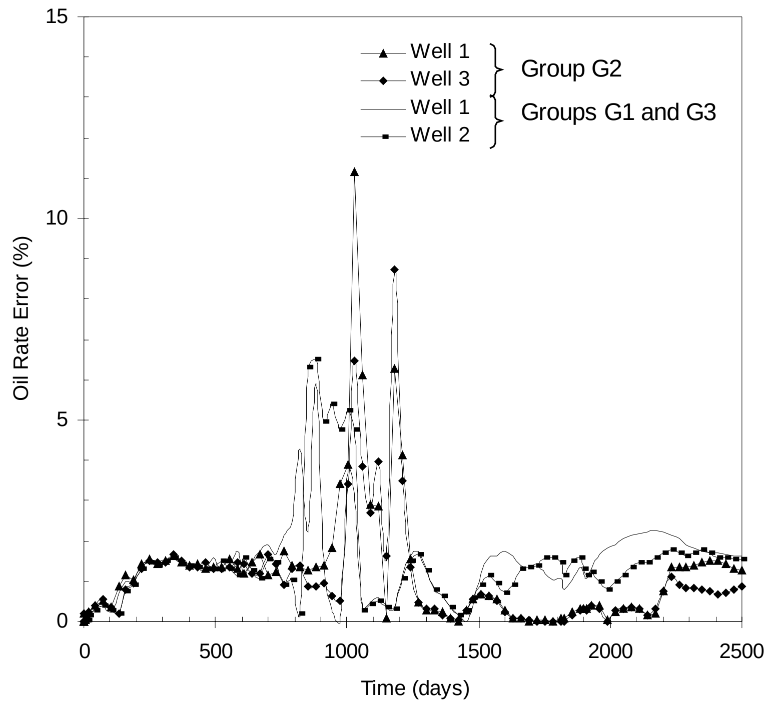


Figure 5.15: Well Oil Rate due to Explicit Treatment of Facilities Model

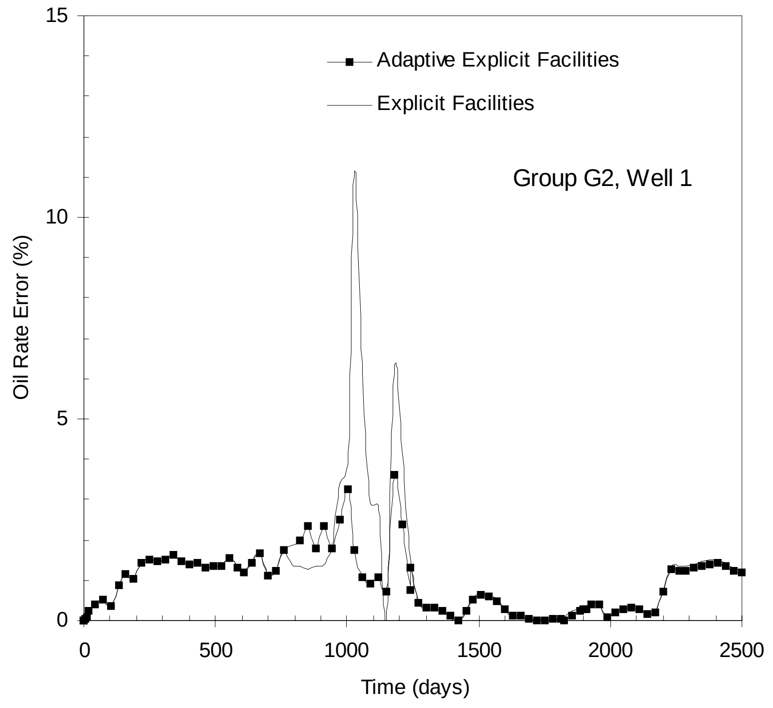


Figure 5.16: Oil Rate Error Comparison – Explicit and Adaptive Explicit Methods

from an explicit to implicit facility formulation when the percent change in solution gas-oil ratio exceeds 1% $\left(\varsigma_{\max} = 1\%\right)$. A comparison of the methods using the rate data for well 1 of group G2 is shown in Figure 5.16. The oil rate error is significantly reduced and has a maximum value of only 4%. The errors for the other wells show similar reductions.

5.6.3 Problem 1 - Computational Cost of Formulations

Nonpreconditioned Formulations

The efficiency of the explicit and adaptive explicit formulations is examined with respect to the standard implicit formulation. The Newton convergence behavior for each formulation is shown in Table 5.7. The affect of facility coupling on the standard implicit formulation is evident. The number of fully coupled iterations exceeded the number of facility domain iterations required by the explicit formulation. In the adaptive explicit formulation the fully coupled iterations reflect the cost of the standard implicit timesteps, while the combination of the reservoir domain and facility domain iterations reflects the cost of the timesteps using the explicit facilities formulation. A comparison of the standard implicit (*fully coupled*) and adaptive explicit (*fully coupled + reservoir domain*) iterations shows a 24% reduction in the number of iterations involving a linear solve that includes the reservoir domain when compared to the standard implicit formulation, which highlights the benefit of the adaptive explicit method

Table 5.7: Performance Data – Nonpreconditioned Formulations – Problem 1

	Newton Iterations			CPU Reduction[1]
Coupled Formulation	Fully Coupled	Reservoir Domain	Facility Domain	
SI-Fac	664	-	-	
Exp-Fac	-	429	550	39%
AE-Fac	168	334	421	33%

[1]reduction is with respect to CPU requirements of standard implicit facilities (SI-Fac) formulation

The explicit formulation reduced the standard implicit CPU requirements 39% by decoupling the reservoir and facility domains which allowed for improved convergence behavior in the reservoir domain, however at the expense of errors of the order 5 to 11% in the well rates (Fig. 5.15). The adaptive explicit formulation performed quite well and reduced the standard implicit CPU requirements 33%, in addition to minimizing the affect of explicit coupling on well rates (Fig. 5.16).

Preconditioned Standard Implicit

The Newton convergence rate of the standard implicit formulation is considerably reduced due to the facility coupling. The preconditioned runs resulted in reduced standard implicit iterations on the fully coupled problem, at the cost of iterations on the preconditioner (Fig. 5.17). The optimal preconditioner configuration for this problem is a 1x1x1 subdomain size for each well. Preconditioning reduced the fully coupled iterations by 31% and resulted 17% reduction in total CPU time. The selection strategy (Section 5.5.1) limited preconditioning considerably for the 3x3x5 case and resulted in an increased number of fully coupled iterations. This behavior may be due to the increased internal boundary area associated with the 3x3x5 subdomains, which can increase the occurrence of poor preconditioned solutions.

The performance of the configurations is summarized in Table 5.8. The difference in savings when comparing iteration and CPU requirements is examined by first establishing the cost of Jacobian, linear solve and preconditioner components of the formulations and then examining the total cost per iteration. Note that the Jacobian and linear solve components are with respect to standard implicit iterations only, while the preconditioner component contains costs for Jacobian and linear solve computations.

The cost per iteration for the Jacobian calculations and linear solve is consistent among the standard implicit and preconditioned cases. The reduced cost for these calculations in the explicit formulation reflects the absence of facility components when solving on the reservoir domain, which increased the overall efficiency of the formulation considerably. The cost per iteration of the preconditioner increases only slightly with subdomain size since the costs of the network calculations dominate this component. The cost of the preconditioner is discussed further in Chapter 6. In the explicit formulation,

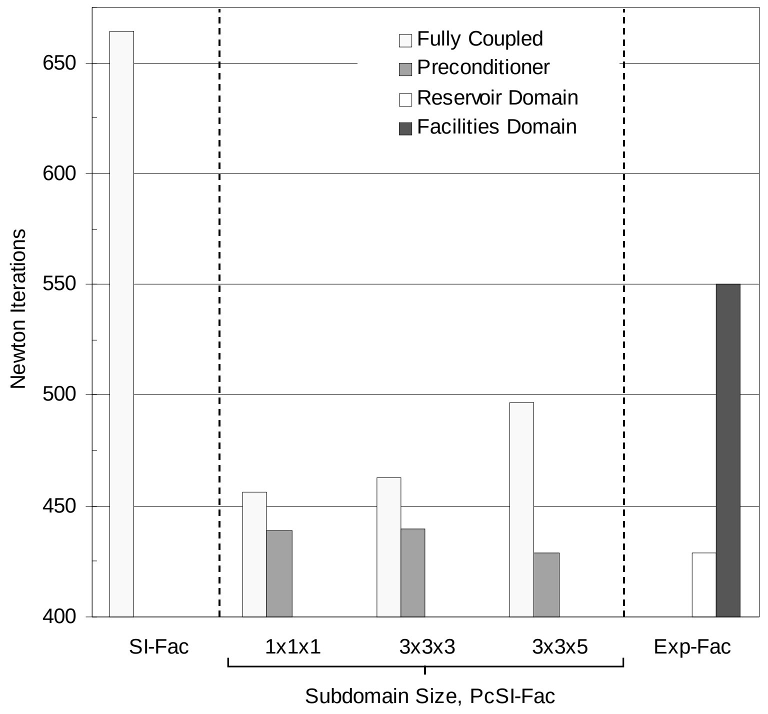


Figure 5.17: Preconditioned Standard Implicit Convergence Behavior – Problem 1

Table 5.8: Preconditioned Standard Implicit Facilities Performance Data – Problem 1

	% Reduction[2]		Cost of Formulation Components (sec/iteration)			
Coupled Formulation[1]	Newton Iterations[3]	Total CPU Time	Jacobian Calculations	Linear Solve	Preconditioner[4]	Total
SI-Fac	-	-	1.44	4.9	-	6.34
PcSI-Fac, 1x1x1	31.3	16.9	1.53	5.18	0.99	7.70
PcSI-Fac, 3x3x3	30.3	14.6	1.52	5.16	1.09	7.77
PcSI-Fac, 3x3x5	25.1	6.8	1.50	5.32	1.67	8.49
Exp-Fac	35.4	38.5	0.79	4.0	0.94	5.73

[1]well subdomain size included with preconditioned formulations
[2]with respect to standard implicit results
[3]depending on formulation, reduction based on fully coupled or reservoir domain iterations
[4]data for explicit formulation corresponds to computation performed in the facility domain

the cost per iteration required for solving the facility domain is consistent with these results.

The average total cost per preconditioned standard implicit iteration is 7.98 seconds/iteration versus 6.34 for a standard implicit iteration. The primary difference is due to the cost of preconditioner which increased the cost per iteration by 27% when compared to a standard implicit iteration, and thus accounts for the difference in the Newton iteration and CPU performance metrics. However, note that even though the net cost per fully coupled iteration is higher for the preconditioned runs, preconditioning reduces the number of standard implicit iterations that require a linear solve on the fully coupled Jacobian matrix and results in a significant savings when compared to the cost of the preconditioner.

Preconditioned Adaptive Explicit Facilities

The results shown in the previous section demonstrate that the adaptive explicit formulation is able to minimize the material balances errors associated with the explicit formulation. Selective application of the standard implicit formulation results in favorable Newton iteration convergence behavior that is further improved through preconditioning. The convergence performance of the method is shown in Fig 5.18 and Table 5.9. To simplify the presentation, the number of iterations shown for the adaptive formulations represents a combination of components. The iterations required for the fully coupled and reservoir domains are combined since their cost is similar. Both iterations require a linear solve based on a Jacobian matrix that includes the full reservoir flow field coefficients. Similarly, the iterations required for the preconditioner and facility domains are combined since their cost is similar due to the pipe network calculations.

The nonpreconditioned adaptive explicit case (AE-Fac) required a total of 502 fully coupled and reservoir domain iterations. The optimal preconditioner configuration for this problem is a 3x3x5 subdomain size for each well and reduced the fully coupled iterations by 14% and resulted 4% reduction in total CPU time. This percentage is lower than the equivalent comparison for the standard implicit and preconditioned standard

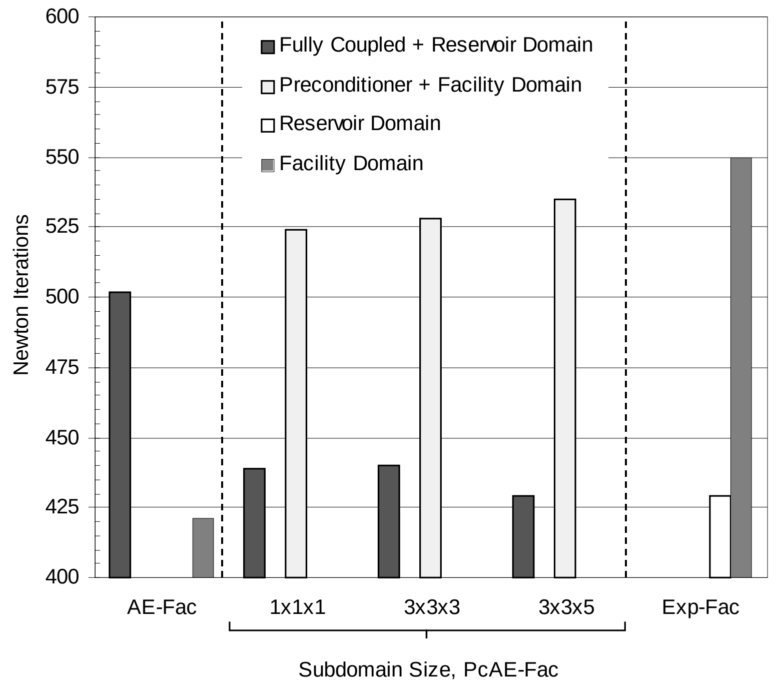


Figure 5.18: Preconditioned Adaptive Explicit Facilities Convergence Behavior

Table 5.9: Performance Data – Adaptive Explicit Facilities Formulation – Problem 1

	% Reduction[1]	
Formulation	Newton Iterations[2]	Total CPU Time
PcAE-Fac, 1x1x1	12.3	2.6
PcAE-Fac, 3x3x3	12.3	2.3
PcAE-Fac, 3x3x5	14.5	4.0

[1]with respect to adaptive explicit facilities (AE-Fac) results
[2]reduction based on fully coupled and reservoir domain iterations

implicit cases (Table 5.8) due to the selective application of the standard implicit formulation, which was applied on only 20% of the timesteps while an explicit facility coupling was applied for the other 80%.

The difference in facility and preconditioner plus facility domain iterations required by the respective nonpreconditioned (AE-Fac) and preconditioned adaptive explicit (PcAE-Fac) formulations are consistent with the strategy design (Fig. 5.18). The facility domain required for the nonpreconditioned case corresponds to the timesteps when the adaptive formulation is in explicit facilities mode. When the preconditioner is activated, the work performed by the fully coupled iterations for converging the facilities is shifted to the preconditioner and accounts for the large increase in preconditioner plus facility domain iterations for the various well subdomain configurations. Note that the iteration requirements for the preconditioned cases (preconditioner + facility domain) are similar to the explicit facilities formulation (facility domain), which is due to the cost facilities Jacobian calculations.

Summary of Results

The results for the explicit facilities and optimal preconditioned standard implicit adaptive explicit facilities configurations are summarized in Table 5.10. The new preconditioning method is shown to improve convergence rate and reduce computational requirements of the base formulations. The preconditioned standard implicit formulation improved the Newton convergence rate considerably, however due to the cost of preconditioning, resulted in a reduction in CPU time of only 16%. The adaptive explicit formulation combined the benefits of improved standard implicit iterations with the efficiencies of the explicit formulation and resulted in a 36% reduction in total CPU time when compared to the standard implicit formulation. Additionally, the method only required 5% more CPU time than the explicit coupling, and minimized the material balance errors when compared to the explicit coupling.

Table 5.10: Comparison of Optimal Preconditioner Configurations – Problem 1

	% Reduction[1]	
Formulation	Newton Iterations[2]	Total CPU Time
Optimal (1x1x1) PcSI-Fac	31.0	17.0
Optimal (3x3x5) PcAE-Fac	35.9	36.2
Exp-Fac	35.9	38.6

[1]with respect to standard implicit results
[2]depending on formulation, reduction based on fully coupled and/or reservoir domain iterations

5.6.4 Problem 2 - Model Description

The second test problem simulates production from three well groups over a four year time period. The model demonstrates the impact of gas breakthrough and well placement on the surface facilities behavior. Additionally, results from a variation of the base model are included to demonstrate the interaction between the reservoir and facilities. The base model and data are shown in Fig. 5.19 and Tables 5.11-13. The rock-fluid and PVT properties are listed in Appendix B. The wells are grouped according to completion interval with a maximum of three wells per group. Similar to Problem 1, an injection group is included for pressure maintenance.

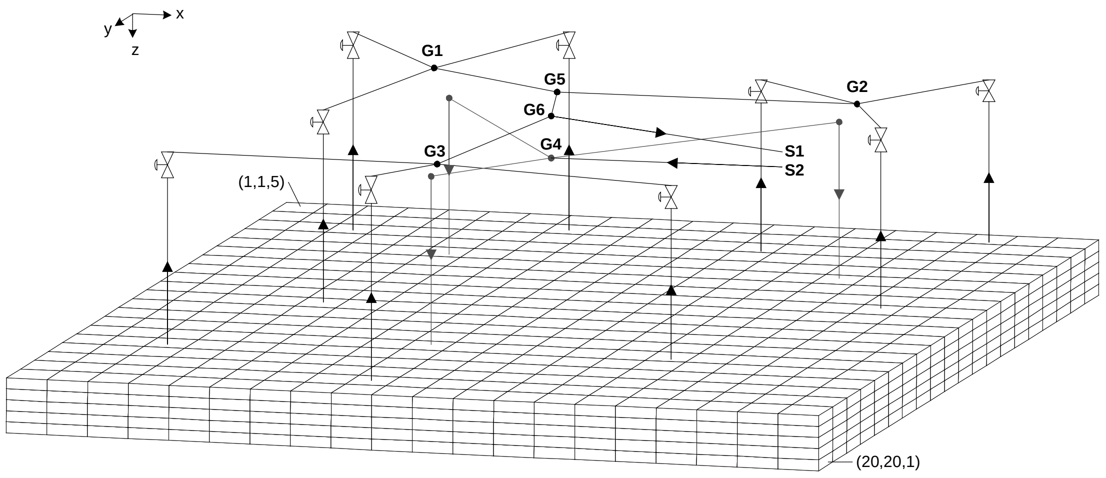


Figure 5.19: Reservoir and Surface Facilities Layout - Test Problem 2

Table 5.11: Reservoir Data – Problem 2

Parameter			
nx,ny,nz	20x20x5	depth to top of reservoir	8337.5’
dx=dy	500’	kx=ky, layer 5	200 md
dz	25’	layer 4	500 md
porosity @ 14.7 psia	0.25	layer 3	200 md
pressure @ 8325’	5800 psia	layer 2	100 md
initial water saturation	0.12	layer 1	500 md
initial oil saturation	0.88	kz	0.1*kx

Table 5.12: Surface Facilities Data – Problem 2

Junction/Source/Sink	Coordinate location[1]		
G1	(2750,2750,0)	G5	(4750,2750,0)
G2	(7750,2250,0)	G6	(5250,4750,0)
G3	(4250,7250,0)	S1	(8200,4750,0)
G4	(5250,4750,0)	S2	(8200,4750,0)

Surface pipe data[2]			
Production Gathering Network	Diameter	Boundary Conditions	
G1 to G5	6.0”	S1	1750.0 psia
G2 to G5	6.0”	S2	5810.0 psia
G5 to G6	6.0”	Choke Parameters[3]	
G3 to G6	6.0”	diameter	0.9”
G6 to S1	8.0”	discharge coef.	0.85
Injection Network			
S2 to G4	6.0”		

[1]reference location is the corner of cell (1,1,5) with coordinates (0,0,0), expressed in units of feet.
[2]absolute roughness of all pipe equals 0.0001’
[3]same for all production wells

Table 5.13: Surface Facilities Data – Problem 2

		Well block	Tubing data[1]		Wellhead to junction node[1]	
Group	Well	(i,j,k)	Vertical length	Diameter	Horizontal length	Diameter
G1	1	(3,3,1)	8450’	4.5”	1767’	4.5”
	2	(8,2,1)	8450’	4.5”	2150’	4.5”
	3	(5,11,1)	8450’	4.5”	2761’	4.5”
G2	1	(13,3,2)	8425’	5.0”	1457’	4.5”
	2	(18,1,2)	8425’	5.0”	2150’	4.5”
	3	(18,9,2)	8425’	5.0”	2573’	4.5”
G3	1	(3,16,,2)	8425’	3.5”	2850’	4.5”
	2	(9,19,2)	8425’	3.5”	2263’	4.5”
	3	(17,15,2)	8425’	3.5”	4257’	4.5”
G4	1	(6,5,5)	8350’	3.0”	3181’	3.0”
	2	(16,6,5)	8350’	3.0”	3259’	3.0”
	3	(9,15,5)	8350’	3.0”	2850’	3.0”

[1]absolute roughness equals 0.0001’

The production period is limited to 4 years due to the surface facilities implementation that does not allow for redefining well membership for a group based on operating conditions. The injection gas reaches group G2 wells earlier than the other two groups and at breakthrough results in loss of pressure support for the G2 wells. These wells would continue to flow only if the downstream pressure decreased enough to compensate for the drop in reservoir pressure. The production from groups G1 and G2 is combined at collection point node G5. Since gas breakthrough at the group G2 wells did not affect production from group G1 wells, pressure support at G5 is unaffected and the group G2 wells cannot flow. Fig 5.20 shows the production profile for well 1 of group G2. The reservoir deliverability curve represents well flowing rate in the absence of hydraulic and frictional pressure drops. The facility capacity curve indicates how much fluid from well 1 can flow into collection point G2, given the tubing and surface pipe performance, and pressure at G2. As shown in Fig. 5.20, well 1 cannot flow for approximately 800 days due to the loss of pressure support and high backpressure at the collection node.

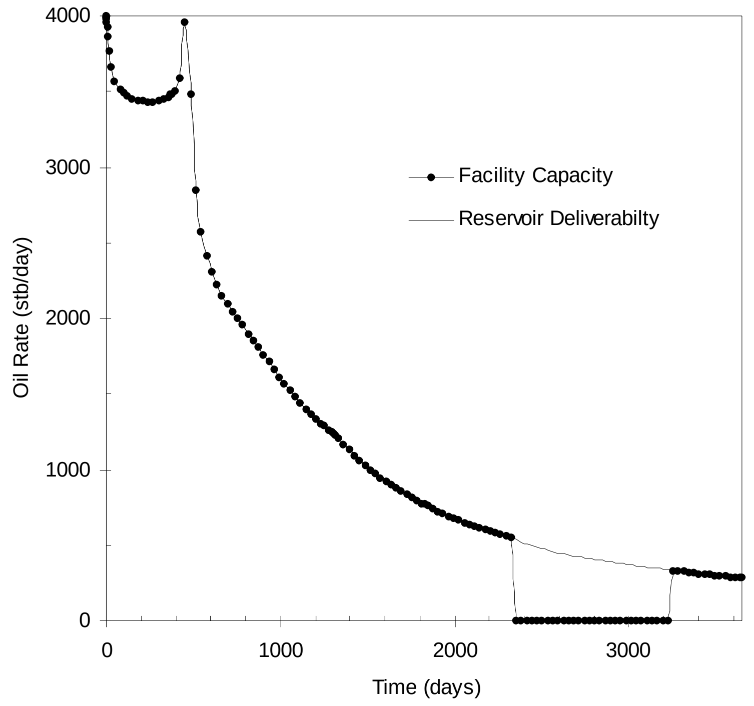


Figure 5.20: Production Profile, Group G2 Well 1

After 3200 days the collection node pressure drops enough so that well 1 is brought back on production. The other two wells in group G2 are removed from production for the remainder of the simulation. In this situation the group G2 wells could have continued to flow if they were moved to a lower pressure gathering system.

The operating objective is to obtain the maximum field production rate where each wellhead choke is fixed at 90% open and the separator pressure for the production gathering system is held constant at 1750 *psia*. The production rates for each group (Fig. 5.21) show that group G1 contributes approximately 60% of the total field rate and the remaining 40% is distributed between groups G2 and G3. For each well the tubing and surface pipe properties are similar and the primary difference in production rates are due to well placement. Wells in group G1 are completed in a higher permeability zone when compared to the permeability zone for wells in groups G1 and G2. Thus for an equivalent pressure drop, group G1 wells flow at a higher rate than the other wells as shown by the differences in production profiles.

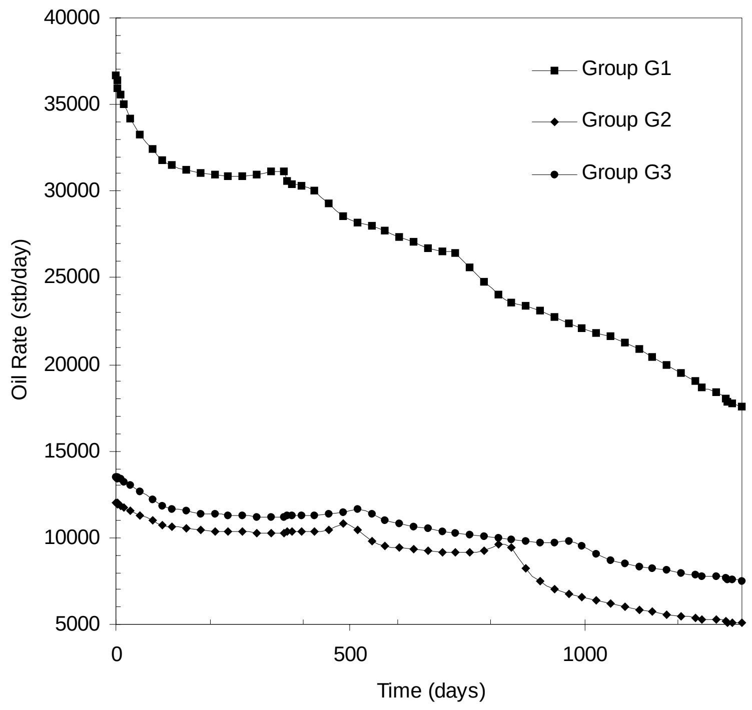


Figure 5.21: Group Oil Rate Production Profiles - Test Problem 2

5.6.5 Problem 2 - Accuracy of Formulations

Explicit Facilities

The performance of the explicit coupling is presented with respect to material balance errors based on the difference between beginning of timestep explicit facility and standard implicit well rates. A similar profile exists when comparing beginning of timestep explicit facility rate versus the end of timestep reservoir rate. As shown in Figs. 5.22-24, several wells incurred material balance errors that exceeded 10%, with a maximum error of 25% for well 1 of group G2. In contrast to the previous test problem, where the development of two-phase flow in the wellbore occurred within a 500 day time interval, the time interval for this problem spans 800 days, with significant variations in individual well behavior.

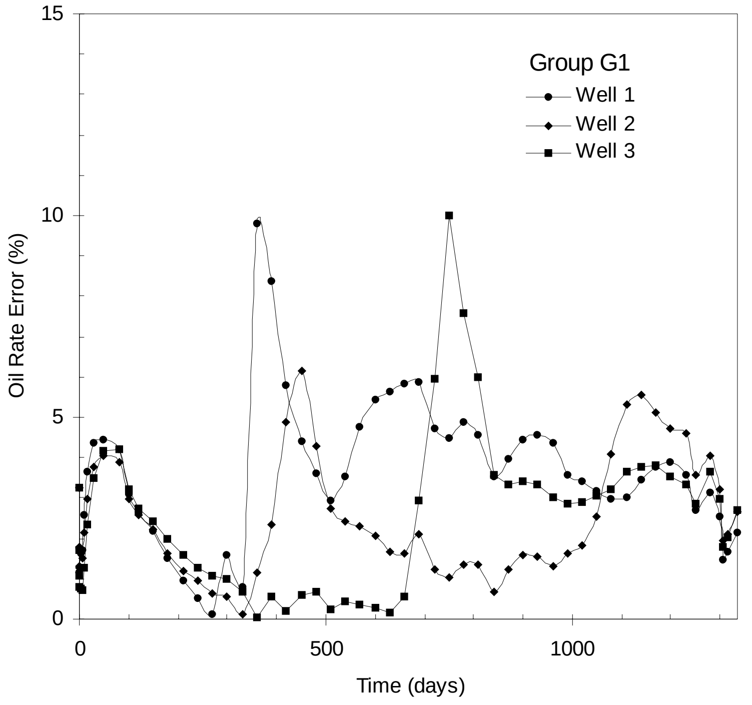


Figure 5.22: Oil Rate Error – Explicit Treatment of Facilities Model – Group G1

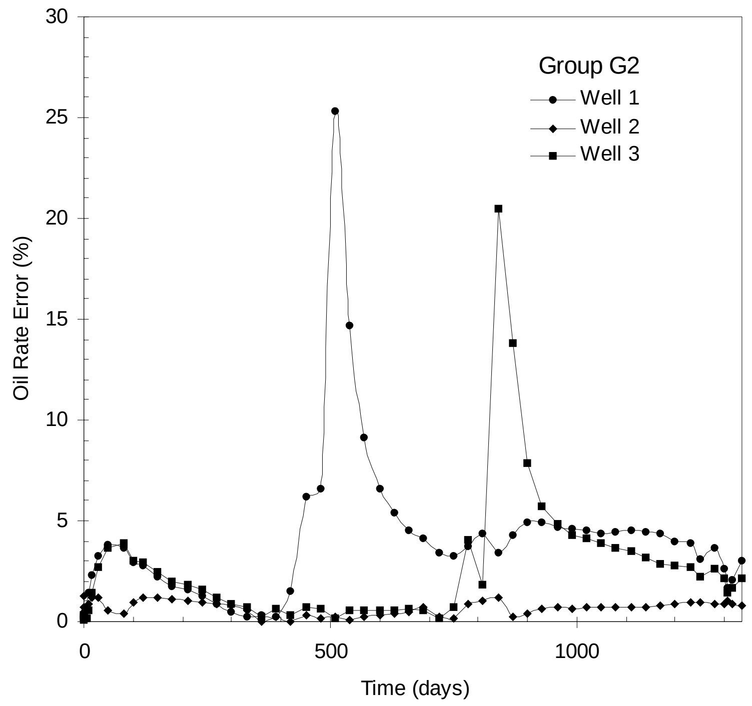


Figure 5.23: Oil Rate Error – Explicit Treatment of Facilities Model – Group G2

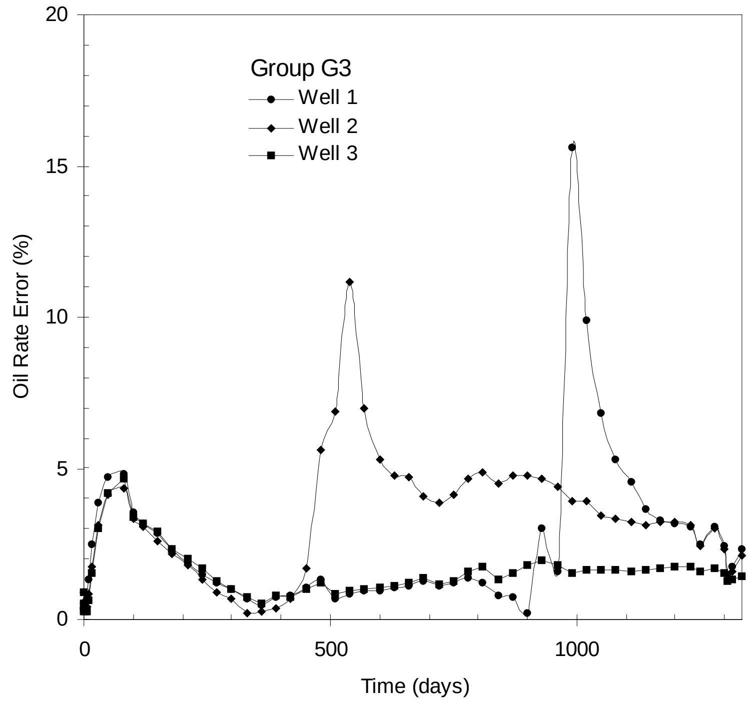


Figure 5.24: Oil Rate Error – Explicit Treatment of Facilities Model – Group G3

Adaptive Explicit Facilities

Similar to Problem 1, the adaptive explicit facilities formulation is employed to exploit the performance benefits of the explicit formulation outside of the transition periods. In this problem, the method switched from an explicit to implicit facility formulation when the percent change in solution gas-oil ratio exceeded 5% ($\varsigma_{\max} = 5\%$). A comparison of the methods using the rate data for well 1 of groups *G1* and *G2* is shown in Figure 5.25. The oil rate error is significantly reduced and has a maximum value of only 3%. The errors for the other wells show similar reductions.

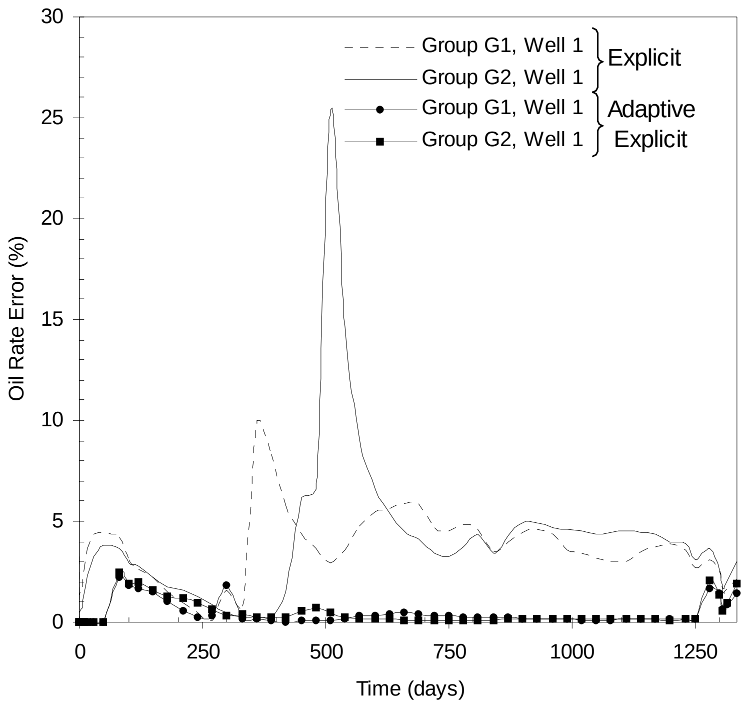


Figure 5.25: Oil Rate Error Comparison – Explicit and Adaptive Explicit Methods

5.6.6 Problem 2 - Computational Cost of Formulations

Nonpreconditioned Formulations

The efficiency of the explicit and adaptive explicit formulations is examined with respect to the standard implicit formulation. The Newton convergence behavior for each formulation is shown in Table 5.14. Similar to Problem 1, the convergence rate of the standard implicit formulation is considerably lower than the rate of the explicit formulation on the reservoir domain and is more similar to the convergence rate of the facility domain component. The explicit formulation reduced the standard implicit CPU requirements 51%, and as shown in Figs. 5.22-24, at the expense of considerable material balance errors on the order of 5 to 25%. The adaptive explicit formulation minimized the affect of explicit coupling on well rate errors, but reduced the standard implicit CPU requirements by only 17%, in contrast to the 33% reduction achieved in Problem 1. This is due primarily to the large time variance corresponding to the development of two-phase flow in the wellbore.

Table 5.14: Performance Data – Nonpreconditioned Formulations – Problem 2

	Newton Iterations			CPU Reduction[1]
Coupled Formulation	Fully Coupled	Reservoir Domain	Facility Domain	
SI-Fac	589	-	-	
Exp-Fac	-	268	534	51%
AE-Fac	471	70	93	17%

[1]reduction is with respect to CPU requirements of standard implicit formulation

Preconditioned Standard Implicit

The preconditioned runs reduced the number of fully coupled iterations significantly (Fig. 5.26). The optimal preconditioner configuration used a 1x1x1 subdomain size for each well and reduced the fully coupled iterations by 47.7% and resulted in 35.5% reduction in total CPU time. The 3x3x3 configuration employed a lower number of preconditioned iterations than the 1x1x1 and 3x3x5 configurations and resulted in a slight increase in the number of fully coupled iterations.

The strategy presented in Section 5.5.2 is used to determine when preconditioning should be applied and in this problem the 25% of the Newton iterations are preconditioned for the 1x1x1 and 3x3x5 configuration, in contrast to only 15% for the 3x3x3 configuration. The selection criteria employed by the strategy detected problems with the 3x3x3 configuration and reduced preconditioning frequency and thereby reduced the costs associated with preconditioning (Fig. 5.26). This compensated for the added cost of fully coupled iterations and resulted in computational requirements that are very similar to the 1x1x1 and 3x3x5 configurations (Table 5.15, avg. % reduction in CPU time equals 35%). This demonstrates the benefit of the strategy selection criteria since optimal selection of subdomain size cannot be easily determined and the preconditioning method must be robust enough to take advantage of arbitrary configurations.

The performance of the configurations is summarized in Table 5.15. The difference in savings when comparing iteration and CPU requirements is examined by first establishing the cost of Jacobian, linear solve and preconditioner components of the formulations (Table 5.15) and then examining the total cost per iteration.

The cost per iteration for the Jacobian calculations and linear solve is consistent for the standard implicit and preconditioned cases. Similar to the previous test problem results, the reduced cost for these calculations in the explicit formulation reflects the absence of the facilities component when solving for the reservoir domain, which increased the overall efficiency of the formulation considerably. The cost per iteration of the preconditioner increases only slightly with subdomain size since the costs of the network calculations dominate this component. The cost per iteration required for solving the facility domain with the explicit formulation is consistent with these results.

The average total cost per iteration of the preconditioned runs is 5.6 seconds/iteration versus 4.81 for the standard implicit case. The difference is due to a combination of the added cost of the preconditioner coupled with slightly improved performance in the linear solve.

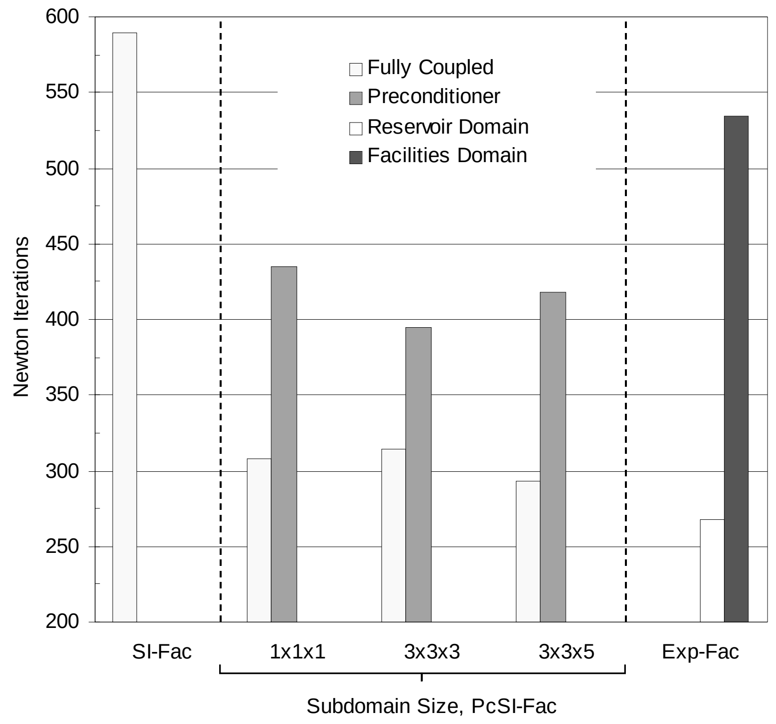


Figure 5.26: Preconditioned Standard Implicit Performance Data – Problem 2

Table 5.15: Performance Data for Preconditioned Formulations – Problem 2

	% Reduction[2]		Cost of Formulation Components (sec/iteration)			
Coupled Formulation[1]	Newton Iterations[3]	CPU Time	Jacobian Calculations	Linear Solve	Preconditioner[4]	Total
SI-Fac	-	-	1.41	3.4	-	4.81
PcSI-Fac, 1x1x1	47.7	35.5	1.53	2.77	1.15	5.54
PcSI-Fac, 3x3x3	46.7	34.0	1.53	2.8	1.28	5.61
PcSI-Fac, 3x3x5	50.2	35.0	1.54	2.83	1.34	5.71
Exp-Fac	54.5	51.0	0.60	2.3	1.13	4.03

[1]well subdomain size included with preconditioned formulations
[2]with respect to standard implicit results
[3]depending on formulation, reduction based on fully coupled or reservoir domain iterations
[4]data for explicit formulation corresponds to computation performed in the facility domain

Preconditioned Adaptive Explicit Facilities

The convergence performance of the preconditioned adaptive explicit method is shown in Fig 5.27 and Table 5.16. The explicit results are included for reference. All preconditioner configurations resulted in very similar performance (Table 5.16). The optimal preconditioned adaptive explicit configuration used a 3x3x3 subdomain size and reduced the number of fully coupled Newton iterations by 45%, which resulted in a 25.6% reduction in CPU time when compared to the nonpreconditioned adaptive explicit formulation. These percentages are higher than the equivalent comparison for the first problem due to the large time variance corresponding to the development of two-phase flow in the wellbore, and thus a higher percentage of timesteps require the standard implicit formulation.

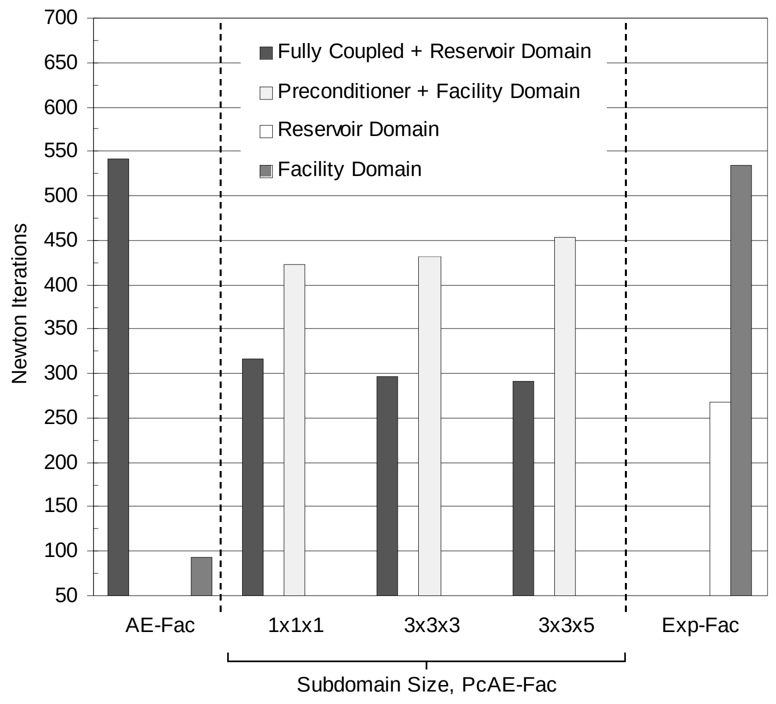


Figure 5.27: Preconditioned Adaptive Explicit Implicit Convergence Behavior

Table 5.16: Performance Data for Adaptive Explicit Formulation – Problem 2

	% Reduction[1]	
Formulation	Newton Iterations[2]	Total CPU Time
PcAE-Fac, 1x1x1	41.4	25.2
PcAE-Fac, 3x3x3	45.1	25.6
PcAE-Fac, 3x3x5	46.0	24.5

[1]with respect to adaptive explicit facilities results
[2]reduction based on fully coupled and reservoir domain iterations

Summary of Results

The optimal configurations for preconditioned standard implicit and adaptive explicit formulations resulted in very similar performance due to the significant number of fully coupled iterations required by this problem. When compared to the standard implicit formulation, the optimal preconditioned adaptive explicit facilities configuration reduced the iterations by 49.5% and corresponds to a 38.4% reduction in CPU time. As shown in Fig. 5.25, the formulation reduced the material balance errors of the explicit formulation considerably. With preconditioning, the method required only 25% more CPU time when compared to the explicit formulation, which is minor when considering the magnitude of the errors.

Table 5.17: Comparison of Optimal Preconditioner Configurations – Problem 2

	% Reduction[1]	
Formulation	Newton Iterations[2]	CPU Time
Optimal (1x1x1) PcSI-Fac	47.0	35.0
Optimal (3x3x3) PcAE-Fac	49.5	38.4
Exp-Fac	54.0	51.0

[1]with respect to standard implicit results
[2]depending on formulation, reduction based on fully coupled and/or reservoir domain iterations

5.7 Concluding Remarks

The following formulations for solving coupled reservoir and surface facilities problems have been investigated:

1. Explicit Facilities
2. Standard Implicit Facilities
3. Adaptive Explicit Facilities
4. Adaptive Implicit Reservoir - Standard Implicit Facilities
5. Preconditioned Standard Implicit Facilities
6. Adaptive Implicit Reservoir - Preconditioned Standard Implicit Facilities
7. Preconditioned Adaptive Explicit Facilities

The methods have been analyzed with respect to the accuracy and convergence characteristics of Newton's method applied to the nonlinear problem. The explicit formulation decouples the reservoir and facility domains and allows for efficient solution of the nonlinear problem at the expense of material balance errors. The standard implicit facilities formulation eliminates the material balance error to a specified tolerance, however the convergence rate is reduced severely and results in a significantly increased CPU requirement. The adaptive implicit reservoir formulation reduces the cost of solving for the reservoir flow field, but similar to the previous formulation, the full facilities coupling results in increased CPU requirements.

An adaptive explicit facilities formulation has been developed which combines the efficiency of the explicit formulation with accuracy close to that of the standard implicit formulation. A new preconditioning method has been developed that can be used with the standard implicit and adaptive explicit facilities formulation to accelerate convergence of the fully coupled iterations. The preconditioning method has been applied successfully to both formulations and test results indicate that the preconditioned adaptive explicit formulation is capable of solving the fully coupled reservoir and facilities problem within an acceptable level of error while reducing the CPU requirements to that of the standard explicit formulation.

The preconditioning technique can also be used with an adaptive implicit treatment of the reservoir flow field. The preconditioner well subdomains are chosen to coincide with the initial boundaries for implicitness, resulting in a preconditioning of the fully coupled adaptive implicit reservoir problem. While not investigated, a formulation which combines the adaptive reservoir flow field and adaptive facility coupling formulations would capitalize on the individual efficiencies of each method. The efficiency of the facility coupling could be further improved through the preconditioning method, resulting in a preconditioned adaptive implicit reservoir adaptive explicit facilities formulation.

Chapter 6

Parallel Formulation

This chapter presents the parallel formulation developed for the coupled reservoir and surface facilities simulation approach. Basic parallel design criteria that motivated the final design are first presented. The key elements of the parallel model are discussed. In addition, the parallel algorithms and code libraries employed from external sources are presented. The overall and specific component parallel efficiencies of the formulation are analyzed based on test model results.

6.1 Parallel Program Design

This section presents the basic criteria and design employed to construct the parallel algorithm. A comprehensive discussion on the design and analysis of parallel programs is provided by Foster (1989).

The design of an efficient parallel algorithm requires analyzing the solution method for a problem to identify components that can be solved concurrently. A component is defined as a unit of parallelism in the algorithm. The component partitioning should contain more tasks than there are processors and should also avoid redundant data operations and storage. Additionally the partitioning should result in a balanced workload among processors and allow for efficient utilization of increased number of processors, commonly referred to as scalability.

The communication costs associated with remote memory accesses must also be considered. A processor can address local data more efficiently than remote data and therefore a design that results in a high frequency of communication will severely restrict individual processor performance. Data sharing among processors also introduces the requirement of methods for managing and executing communication. Statically defined communication patterns between processors simplify the code structure, but may not be appropriate for the problem. In contrast, when dynamically defined communication patterns are required, the code complexity can increase substantially.

Communication between two processors can be executed in two contexts, blocking or nonblocking. A blocking communication operation stops further program execution until the sending and receiving of data is completed. Nonblocking communication operations allow for a send or receive operation to be started while the program execution continues. These contexts have considerable impact on the program efficiency and integrity. Blocking operations can restrict processor execution unnecessarily while nonblocking operations require additional coding effort to ensure data integrity.

6.1.1 Model Design

The problem of coupled reservoir and surface facilities modeling requires the solution of a set of nonlinear partial differential equations. As discussed in Sections 5.2 and 5.4, the solution algorithm for the nonlinear problem and new preconditioning method employs Newton’s method. This method involves the same set of tasks for each Newton iteration, which can be broadly classified as Jacobian generation and the solution of a linear system of equations. Each of these components contains operations that can be performed concurrently.

Concurrency within the linear solve is not considered since an external software package (PETSc) is utilized for this component. A brief discussion of the PETSc package is included towards the end of this section. The basic model for concurrency in the Jacobian calculations is presented next and specific component designs are presented in Sections 6.1.2 and 6.1.3.

Concurrency - Jacobian Calculations

To exploit the maximum concurrency possible, a task for each grid and facilities node can define the decomposition. The task stores the state information of the node and is responsible for computations required for the Jacobian generation process. The finite difference stencil used for the reservoir grid and network connectivity in the facilities model requires that each task obtain values from the tasks responsible for neighboring nodes. Hence there are a total of $n_{xyz} + n_f$ (total reservoir cells + total facility nodes) tasks with data and computation per timestep, each requiring communication with several neighboring tasks. This fine-grained decomposition of the coupled model results in a number of tasks that is several orders of magnitude larger than the number of processors, which coupled with the communication requirements for each node renders this type of partitioning inefficient.

The communication requirement between neighboring nodes suggests that a coarsening of the task data and computations will improve efficiencies together with reducing communication. The calculations required by the reservoir and surface facility nodes are based on different numerical methods. Therefore coarsening criteria that result in a balanced distribution of the computation must take into account the differences in reservoir and facilities based calculations.

Two basic approaches that seek to balance the computational load can be considered. The first approach considered here treats the facilities components as one computational task and then decomposes the reservoir domain to achieve an equivalent distribution of the computational load. The selection of the component distribution requires performance metrics for the respective reservoir and facilities components. In this approach the reservoir and facility domain components are performed concurrently. For example, reservoir Jacobian calculations decomposed corresponding to reservoir layers could lead to global partitioning where one processor is responsible for computations on both reservoir and facilities domains.

The second approach broadly decomposes the Jacobian calculations based on numerical formulation and is a natural strategy for preconditioned adaptive explicit coupling. The reservoir domain calculations are partitioned and performed concurrently, independent of the facilities domain partitioning and processing. Thus in this approach the concurrency is achieved in stages and seeks to achieve load balancing through partitioning and concurrent processing of similar calculations. Note that this approach is more general than the first and can accommodate various load balancing strategies within the high level decompositions.

In this research the latter approach was employed since it reduces program complexity and facilitates code reuse. By decomposing the Jacobian generation tasks into reservoir and facilities domain components, much of the code designed for concurrent processing required by the standard implicit formulation can be reused for concurrent processing in the new preconditioning method. Concurrency in the preconditioning method is discussed in Section 6.1.4.

Parallel Communication

The methods for parallel communication are similar for the reservoir and facilities domain components. In the reservoir domain communication patterns between processors form a regular structure based on grid layout, which in this research, does not change over time. Similarly, the partitioning employed for the facilities domain results in communication patterns that are structured in the context that a facilities node always communicates with same node(s), however the order of communication is a function of the machine load and the computational requirements of individual components. Dynamic structured partitioning of the reservoir problem does not complicate the communication patterns, however there is an additional complexity due to the code required for sending, receiving and merging of partitions. The facilities domain problem is relatively small and a copy of the entire facilities domain problem resides on each processor. This data layout coupled with the model for achieving concurrency in the facilities domain does affect the communication patterns when the facilities problem partitioning is changed.

For the reservoir and facilities domain, data communication between processors employs functions provided by the Message Passing Interface (MPI) based on the

Argonne National Laboratory and Mississippi State University implementation. A detailed description of MPI is presented in Gropp *et al*. (1994). To simplify the program code, blocking operations are employed that require syncronization among all nodes before executation can continue, which limit the efficiency of the algorithm. Recommendations are presented in Chapter 8 for removing these limitations.

Concurrency - Parallel Linear Solve

Parallel linear equation solution methods continue to be an area of active research and we have chosen to rely on public domain codes for this component of the parallel formulation. The Portable, Extensible Toolkit for Scientific Computation (PETSc) public domain package developed at Argonne National Laboratory (Baley *et al*. 1997, 1999) was selected due to the ease of integration and availability of user support. The Generalized Minimal Residual (GMRES) method with parallel block Jacobi preconditioning option was employed for all the test runs presented in this chapter. The block Jacobi preconditioning provides a convenient method for constructing a parallel linear solve, however as the number of processors increases, the quality of the preconditioner decreases and the overall solver performance decreases. Saad (1995) provides a detailed discussion of iterative methods for solving linear systems.

6.1.2 Reservoir Domain

In the reservoir domain, the specific partitioning of the Jacobian calculations is motivated by several factors, which are based on the premise of uniform CPU performance among all the nodes. The size of the problem coupled with individual processor data storage limitations requires a partitioning based on the number of reservoir cells. Since Jacobian calculations require only local cell information, the partitioning is defined by a contiguous and equal distribution of reservoir cells. In this research the partitioning of the reservoir cells is obtained via a layered decomposition of the reservoir and simplifies the parallel linear solver setup. Additionally, to achieve better data locality for the internal domain interface flux calculations, the decomposition is defined so that each internal boundary contains one overlapping layer. However, this requires additional message passing at the end of every Newton iteration for updating the state of the overlapping cells and increases the processor workload.

6.1.3 Facilities Domain

The Jacobian calculations for the surface facilities components require a different approach than described previously, which is due primarily to the defined unit of parallelism coupled with the order dependence of the surface facility calculations discussed in Section 4.4.1. The unit of parallelism is defined as the collection of tasks that can be decoupled from the global problem and solved independently. For example in the reservoir domain, the number of layers assigned to each processing node defines the unit of parallelism. In the facility domain the unit of parallelism is defined to be all computations between either a well and junction node, or between two junction nodes. The order dependence of the flow calculations motivates this partitioning and is examined further by considering the pipe network shown in Fig. 6.1.

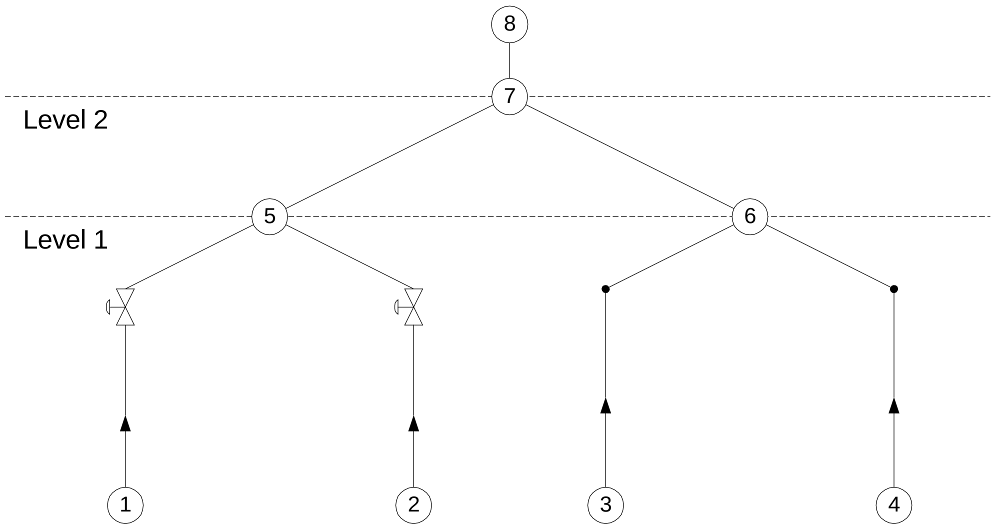


Figure 6.1: Order Dependence in Network Calculations

In this facilities configuration there are two levels of parallelism. The computations that are performed first correspond to level one, which is defined by the connections 1-5, 2-5, 3-6, and 4-6. Using current pressure solutions for each node Jacobian calculations for each of these connections can be performed concurrently. Similarly the calculations for the level two connections, 5-7 and 6-7, can be performed concurrently once the respective calculations upstream of nodes 5 and 7 are completed. Note that the CPU requirement for each connection is a function of the device configuration and phase behavior between the upstream and downstream nodes of the connection. This example also demonstrates how the parallel content is constrained by the surface facility configuration, in addition to the defined unit of parallelism. The first level of parallelism requires at most four processors, and the second level requires at most only two.

The order dependency in the facilities calculations requires a method for task management. Two models for task management are investigated and are referred to as the worker-worker and the manager-worker model. Both models designate one processor, referred to as the manager, for task and data management. The other processors are referred to as workers and perform computation-based tasks. Each model is described further here.

Worker - Worker

The worker-worker model is characterized by the requirement that all processors are candidates for computational tasks, including the manager. Therefore the manager requires a scheme for balancing task and data management with computational assignments. A simple scheme is employed where the manager initially assigns a task to all the workers and then operates in an expanded capacity. The manager performs one computational task and then checks to see if any workers have completed their tasks. Once a worker signals completion, the manager receives the results and allocates another task to the worker. At this stage the manager performs another computational task. The operation sequence for the managers expanded role is summarized as *one local computation –> one receive –> one allocate –> one local computation.*

This scheme does not contain any criteria for balancing the task assignments with respect to computational cost. When the manager performs a computational task, several workers may be idle, waiting to send their results to the manager. As the number of workers increase, the parallel efficiency can be severely restricted since many workers are idle, only for the benefit of the manager's one computational task.

Manager-Worker

The second model employs a scheme designed to maximize the worker's computation time. The manager's role is restricted to task and data management so that the workers' wait time is only limited by the amount of time the manager spends allocating or receiving other respective workers tasks or data. However, since the manager does not perform any computational tasks, the scheme reduces the maximum parallel efficiency by dividing the total amount of computational work over a subset of the available processors.

6.1.4 Newton Step Preconditioner

The preconditioning method developed in Section 5.4 requires construction of a nonlinear problem defined by a collection of subdomains, one for each well, where each well is fully coupled to the facilities domain. The strategies described in Section 6.1.3 for the facilities domain are also employed for parallel implementation of the preconditioner. Since the preconditioner formulation employs fixed size subdomains, the unit of parallelism for the reservoir Jacobian calculations is defined to be a subdomain. Therefore each processor is assigned one or more subdomains so that the load imbalance is minimized with respect to the unit of parallelism. Additionally, since the subdomains are not allowed to overlap, there is no communication required between the subdomains. The following section presents a test problem that is used to analyze the performance of the decomposition schemes.

6.2 Test Results

The performance of the parallel formulations employed for the Jacobian calculations is analyzed here. Linear solver performance data are included for reference. The results are based on data obtained using the coupled model defined in Section 6.2.1. The parallel computing platform used is a Silicon Graphics Power Challenge with 18 R8000 processors and two gigabytes of shared memory. However, the test results are based on runs using only up to 12 processors. The MPI libraries were configured to use shared memory for fast message passing and all codes were compiled with the highest level of optimization available.

6.2.1 Problem Description

The test problem employed simulates production from a 32,000 cell reservoir model. 40 wells are assigned to five production groups and are placed uniformly within the field. The production gathering system is designed so that identical production profiles are obtained for groups G1 and G5. A similar relationship exists for groups G2 and G4. The surface facilities layout and reservoir data are shown in Fig. 6.2 and Tables 6.1-2. The reservoir rock-fluid and PVT properties are listed in Appendix B.

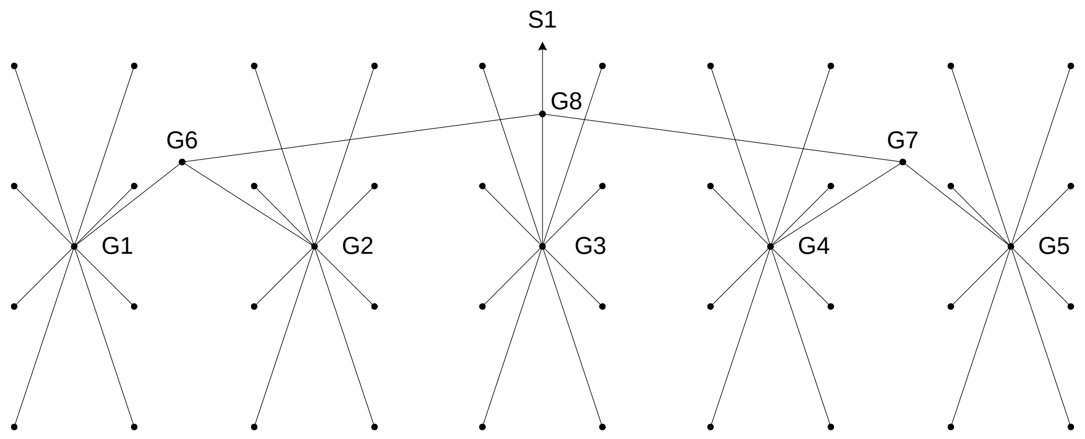


Figure 6.2: Surface Facilities Production Gathering System - Parallel Test Problem

Table 6.1: Model Data – Parallel Test Problem

Parameter			
nx,ny,nz	50x20x32	depth to top of reservoir	4260’
dx=dy	500’	kx=ky	300 md
dz	30’	kz	100 md
connate water saturation	0.12	porosity @ 14.7 psia	0.30
initial oil saturation	0.88	pressure @ 4260’	3000 psia

Table 6.2: Model Data – Parallel Test Problem

Group	Well[1,2,3]	Cell (i,j,k)	Wellhead to junction node[3]	Well[1,2,3]	Cell (i,j,k)	Wellhead to junction node[3]
G1	1	(3,3,1)	5	5	(8,13,1)	3.5”
	2	(8,3,1)	6	6	(3,18,1)	3.5”
	3	(8,8,1)	7	7	(3,18,1)	3.5”
	4	(3,13,1)	8	8	(8,18,1)	3.5”

Junction/Source/Sink	Coordinate location[4]		
G1	(2500,5000,0)	G6	(5000,2500,0)
G2	(7500,5000,0)	G7	(20000,2500,0)
G3	(12500,5000,0)	G8	(12500,1000,0)
G4	(17500,5000,0)	S1	(12500,1000,0)
G5	(22500,5000,0)		

Surface pipe data[3]	Diameter		Diameter
G1 to G6	4.5”	G6 to G8	5.5”
G2 to G6	4.5”	G3 to G8	5.5”
G4 to G7	4.5”	G3 to G8	5.5”
G5 to G7	4.5”	G8 to S1	8.5”
Boundary Conditions			
S1	1500.0 psi		

[1]remaining wells are groups in similar pattern with respect to 10x20 areal section
[2]vertical depth (=5335’) and tubing diameter (=3.5”) is the same for all wells
[3]absolute pipe is roughness 0.0001’
[4]reference location is the corner of cell 1,1,3 with coordinates (0,0,4260), expressed in units of feet.

6.2.2 Performance Results

The methods for parallel reservoir and facilities domain Jacobian calculations are analyzed with respect to results obtained from solving the test model using the preconditioned standard implicit formulation presented in Section 5.3. The 40 wells establish 40 subdomains for preconditioned standard implicit formulation. The well subdomain size for all wells is 3x3x3. The preconditioned method reduced the number of Newton iterations by 34% and resulted in a CPU time reduction of 30% when compared to the nonpreconditioned results.

Performance Metrics

The analysis is based on scalability for a fixed problem size and two performance metrics are used to compare the results. The first metric is the absolute efficiency, $e_{p,a}$, and is defined as follows:

$$e_{p,a} = \frac{T_{seq}}{n_{cpu} T_{n_{cpu}}} \tag{6.1}$$

where T_{seq} is the CPU time required for the sequential run (using one processor), $T_{n_{cpu}}$ is the time required when using n_{cpu} processors. When $e_{p,a} = 1$ the formulation is considered to be 100% efficient. A value of $e_{p,a} < 1$ indicates that the formulation does not scale with the number of processors. The metric $e_{p,a}$ is referred to as “absolute” to indicate that T_{seq} is from the “best” available sequential algorithms, which may not correspond to the “best” available parallel algorithms. This metric constitutes a measure of absolute merit. For example in the following results, the linear solver configuration is not identical for any of the runs and therefore does not permit a comparison of the sequential formulation scalability. If the results are based on state of the art sequential and parallel solvers, then the results would be of value in determining the merit of investing in parallel architectures, assuming speed was the primary concern.

The second metric is the relative efficiency, $e_{p,r}$, and is defined similar to Eq. 6.1, however $T_{n_{cpu}}$ is defined to be the CPU time based on algorithms that are equivalent to those used by the sequential run. This metric characterizes the effectiveness with which a parallel algorithm uses computer resources, independent of problem size. Some algorithms are not designed to achieve 100% efficiency and in those cases an appropriate performance model is required to analyze the efficiency of the formulation. For example, when evaluating the performance of the manager-worker formulation for facilities Jacobian calculations it is important to remember that the parallel model is not designed to achieve 100% parallel efficiency since the manager of the processor does not perform any computation. Therefore the performance of the formulation is analyzed using the following efficiency model:

$$e_p = \frac{T_{seq}}{\left(n_{cpu} - 1\right) T_{n_{cpu-1}}} \tag{6.2}$$

Parallel Efficiency – Overall and Major Solution Components

The parallel efficiency is analyzed for 2, 4, 6, 8 and 12 processor runs, with the uniprocessor run providing baseline data. The overall and component absolute parallel efficiencies are shown in Fig. 6.3. The overall efficiency ranged from 97 to 56% with the best and worst efficiencies corresponding respectively to the 2 and 12 processor configurations. The percent total CPU time spent in each component for the sequential run is also included in Fig. 6.3. The overall parallel efficiency is influenced primarily by the linear solver, which accounts for 72% of the total run time. The reservoir and facilities domain data represents work done outside the preconditioner. The overall efficiency is examined further with respect to the component operations.

Reservoir Domain

The parallel efficiency of the reservoir domain calculations ranged from 95 to 78% for the 2 through 12 processor configurations (Fig 6.3). The reservoir decomposition for each configuration is shown in Table 6.3. The decrease in parallel efficiency is due to the additional cost of calculations for the overlap layers. However, the reduction is considerably less than is to be expected based on percent increase in workload data

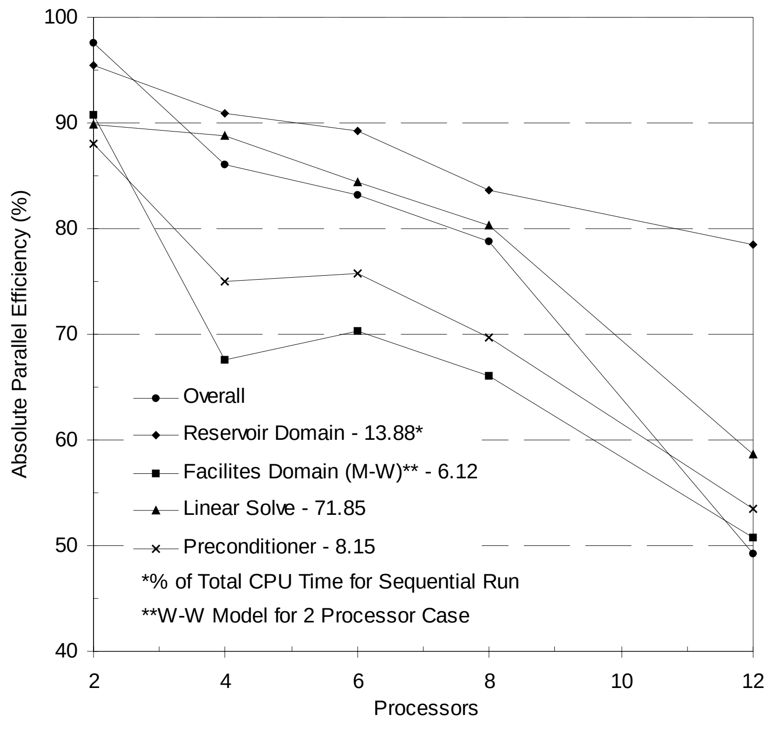


Figure 6.3: Absolute Parallel Efficiencies – Global Solution Procedure Components

Table 6.3: Layered Reservoir Decomposition

	Layers Assigned to Each CPU – without Overlap												Maximum % Increase with Overlap[1]
CPU #	1	2	3	4	5	6	7	8	9	10	11	12	
n_{cpu}													
2	16	16											6.25
4	8	8	8	8									25
6	6	6	5	5	5	5							40
8	4	4	4	4	4	4	4	4					50
12	3	3	3	3	3	3	3	3	2	2	2	2	100

[1]equals number of overlap layers divided by number of non-overlap layers

shown in Table 6.3. For example, the parallel efficiency of the reservoir domain calculations for 12 processor run is 78%. However, the workload for processors 9 through 11 increases by 100%. This indicates that other factors are influencing these results. One possibility is that as the size of each reservoir domain decreases the computational performance of the processor increases due to better data caching which offsets the increased workload.

Facilities Domain

The parallel efficiency of the facilities domain calculations ranged from 94 to 50%. Except for the two processor results, the results for the facilities domain calculations are based on the manager-worker parallel model. The profile for facilities domain and preconditioner efficiencies is similar and allows for analysis of the facilities domain calculations to be included with the analysis of the preconditioner component.

Preconditioner

The parallel efficiencies for the preconditioner components are shown in Fig. 6.4 and highlight several performance features of the method. The efficiency of the linear solve is limited due to the size of the linear problem. As the number of processors increases the efficiency decreases more rapidly than in the linear solve for the global problem (Fig 6.3). Also, note that the linear solver only accounted for 3.5% of the total preconditioner time.

The preconditioner subdomain calculations scaled perfectly and accounted for 12% of the total preconditioner time. The results are consistent with the workload distributions for each case shown in Table 6.4. In several of the cases the 40 subdomains are equally distributed and in the remaining cases the allocation variance is minimal.

Since facilities calculations accounted for 84% of the total preconditioner CPU time, the overall preconditioner and facilities domain efficiencies are nearly identical. Except for the two processor results, the results for the facilities domain calculations are based on the manager-worker parallel model. In general this model performed better than the worker-worker model. An analysis of the worker-worker and manager-worker follows.

Worker-Worker Model

The intent of the worker-worker model is to allow for all processors to participate in the computational work, with a designated manager processor assigned the additional responsibility of task and data management. The performance of this model is examined by comparing the individual processor performance for the two and four processor cases.

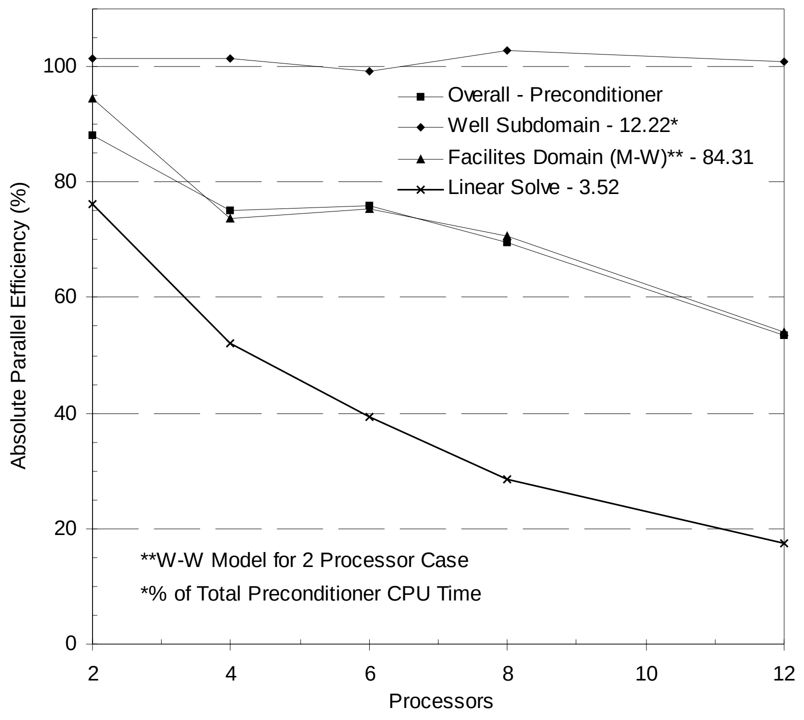


Figure 6.4: Absolute Parallel Efficiencies for Preconditioner Components

Table 6.4: Preconditioner Decomposition

	Number of Subdomains Assigned to Each CPU												Distribution Variance
CPU #	1	2	3	4	5	6	7	8	9	10	11	12	
n_{cpu}													
2	20	20											0
4	10	10	10	10									0
6	7	7	7	7	6	6							0.22
8	5	5	5	5	5	5	5	5					0
12	4	4	4	4	3	3	3	3	3	3	3	3	0.22

Processor Utilization

The processor utilization measures the amount of work performed by an individual processor with respect to the total amount of work required for a task. The individual processor utilization is compared with respect to the distribution of the time spent on computational tasks (Jacobian calculations) and waiting for tasks, or in the case of the manager, the time spent waiting to assign tasks. The processor utilization *u*, is defined by

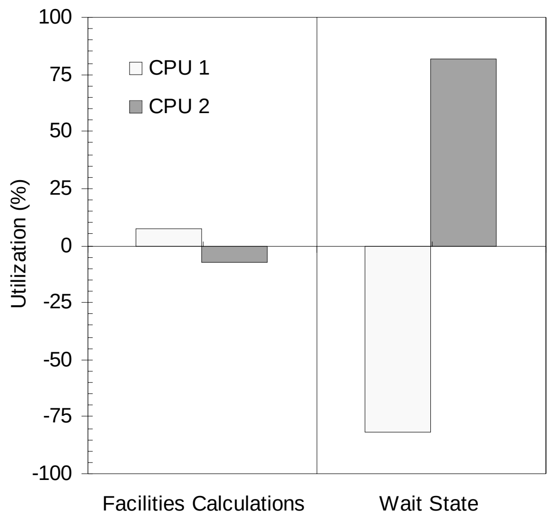
$$u_{j,o} = \frac{t_{j,o} - \overline{t_o}}{\overline{t_o}} \tag{6.3}$$

where $t_{j,o}$ is the total time spent by processor *j* on an operation *o* (computational, wait state/task assignment) and $\overline{t_o}$ is the average time spent on an operation when considering all processors. Note that in this definition an optimal allocation is shown by $u_{j,o} = 0$ for all processors and operations. In the case of a *computational* operation, if $u_{j,o} > 0$, then the processor was assigned a higher fraction of the total computation time required for that operation when compared to the other processor's assignments. If the operation is a wait state, then $u_{j,o} > 0$ indicates that the processor spent more time waiting on assignments than other processors. Similarly, if $u_{j,o} < 0$, then the processor was assigned a lower fraction of the total time required for that operation when compared to the other processor's assignments. The larger the discrepancy among individual processor utilization, the more inefficient is the allocation model. An alternative measure could have been the variance of the time spent for each operation, however much of the model performance is abstracted from the analysis of load balance.

The first and second charts in Fig. 6.5 show the processor utilization for the respective two and four processor configurations. The last processor for both cases is designated as the manager. The two processor configuration is examined first. Recall that parallel efficiency of the worker-worker model for the two processor case is 94% (Fig. 6.4, Facilities Domain). The difference in processor utilization for the computational based tasks is minor (< 4%) and accounts the parallel efficiency (data not shown). The total time spent by each processor in a wait state is less than 3% of the total time spent on facilities calculations. The large wait state time for the manager (processor 1) reflects the cost of blocking send and receive operations. As discussed in Section 6.1.2 the sequence of operations conducted by this processor is:

one local computation –> one receive –> one allocate –> one local computation.

The manager completes a local computation and cannot continue further computation until a worker process sends data. If nonblocking operations were employed the manager process would query worker processes for task completion and continue operation if no worker is ready to send results. However this functionality complicates the code implementation and therefore was not implemented.



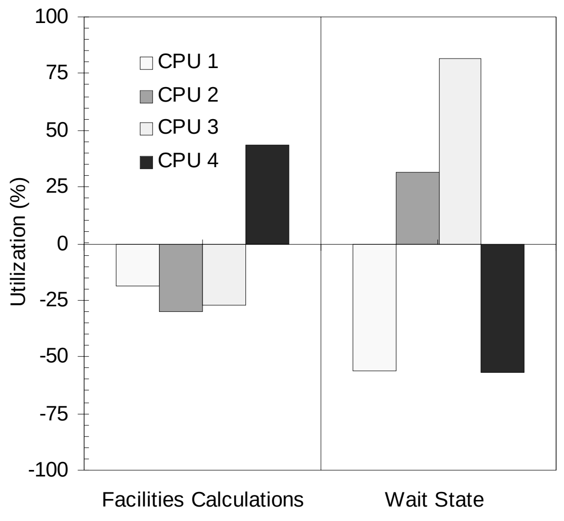


Figure 6.5: Processor Utilization – W-W Model – 2 and 4 CPU Cases

The parallel efficiency of the facilities calculations for the four processor configurations is only 52%. Examination of the second chart in Fig 6.5 shows that the manager took over a significant percentage of the computation load. For every three tasks sent to the workers, the manager would perform three tasks, thereby reducing the efficiency of the remaining processors. The wait state data reflects the order of task assignment by the manager. The manager checks for task completion and performs task assignment in order of increasing processor rank. Thus processor 1 spends considerably less time in a wait state than processor 3.

The results based on configurations with more than four processors demonstrated similar limitations as the four processor case. The manager's strategy for balancing task assignment with individual computation tasks severely limited the parallel efficiency beyond two processors. The efficiency of the manager-work model is examined next.

Manager-Worker Model

The absolute parallel efficiencies of the facilities domain calculations ranged from 75 to 50% (Fig. 6.3) and reflect a reduced efficiency due to the role of the manager. However, when the parallel efficiency of the formulation is examined via the efficiency model shown in Eq. 6.2, the manager-worker strategy is shown (Fig. 6.6) to perform as designed for four and six processor configurations, and resulted in respective parallel efficiency of 98 and 92%. An analysis of the individual processor performance of the four processor configuration reinforces these results. The eight processor configuration is included to complete the analysis. The load and wait state utilization for these configurations are shown in Fig. 6.7. The computational load for the three processors is nearly identical and accounts for the 98% relative efficiency. In this configuration the manager was able to keep the workers operating at peak efficiency. A similar behavior is observed for the six processor configuration. However, in the eight processor configuration, the computational load imbalance reduced the relative parallel efficiency of the formulation to 81%. This result is quite significant because it indicates that even with a dedicated manager, as the number of workers increase, the manager is unable to effectively allocate tasks. The wait state load utilization shows this result through the high percent of wait state time associated with the higher rank processors.

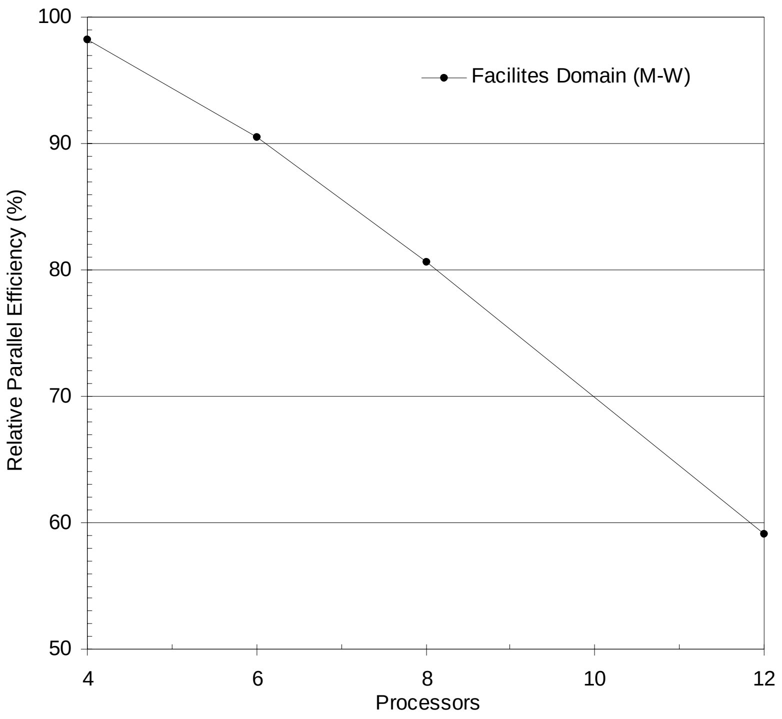


Figure 6.6: Relative Parallel Efficiencies – Facilities Domain – M/W Model

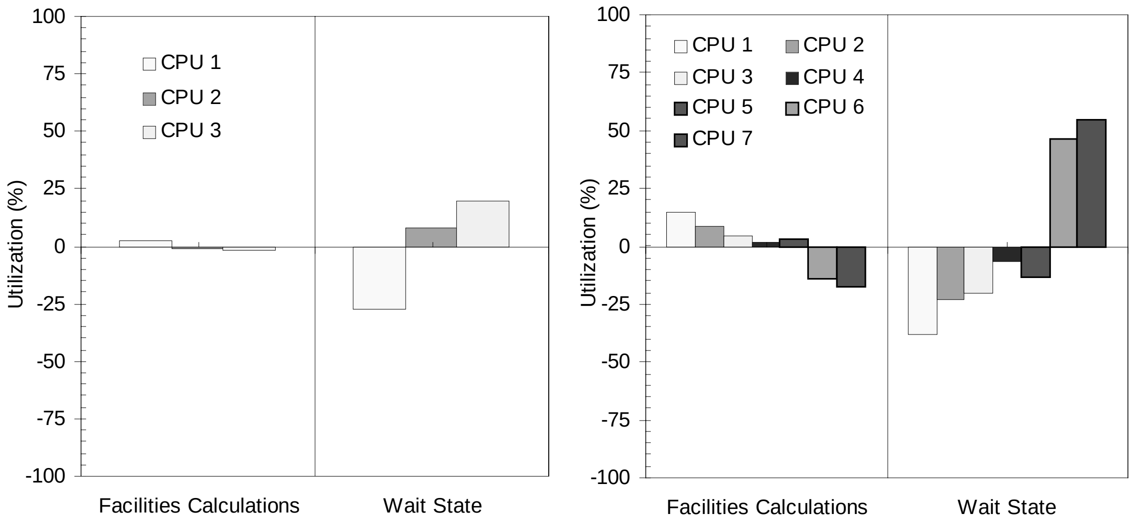


Figure 6.7: Processor Utilization – M-W Model – 2 and 8 CPU Cases

6.3 Concluding Remarks

The parallel strategies applied here have been developed for the Jacobian calculations for the reservoir, facilities domains and Newton preconditioner. The decomposition of the reservoir domain is motivated by problem size. The overlap layers at the internal domain boundary interfaces influence the parallel efficiency. The well subdomains require similar reservoir calculations in the preconditioner. The computations required for the subdomains are distributed among all the processors so that the load imbalance with respect to the number of subdomains assigned to one processor is minimized. This allocation scheme was shown to scale very well since the subdomains are not overlapping. In addition to the subdomain components, the parallel formulation for the preconditioner contained facility domain components, which influenced the overall parallel efficiency of the preconditioner strongly.

The decomposition of the facilities domain is motivated by the network configuration and order dependency of the calculations. Two parallel models were investigated, the worker-worker and manager-worker. The worker-worker model distributed the computational tasks among all processors. One of the processors is designated as the manager and also performed task management operations. The manager’s strategy for balancing task assignment with local computation severely restricted parallel efficiency beyond two processors.

The manager-worker model is designed to maximize the computation efficiency of the worker processors at the expense of limiting the manager’s responsibility to task management. Therefore the overall parallel efficiency is restricted by the parallel model, however it was shown that given a facility problem with 40 wells, the efficiency of the implementation was consistent with the parallel model’s peak performance capability for up to six processors. However, for processor configurations greater than six the efficiency of the formulation degraded and indicated that even with a dedicated processor for task management, the efficiency of the worker processors can be limited.

The combination of the results from the worker-worker and manager-worker model are used to develop the recommendations for further improving the parallel efficiency of the facilities calculations and are presented in Chapter 8.

Chapter 7

Object-Oriented Design

This chapter focuses on the design and development of the application code used for this research. The design is based on object-oriented techniques and the application is developed with the C++ programming language. The framework of the code is presented in terms of object and computational domain models. Each of these models is discussed in detail and several examples are presented to demonstrate the reuse and extendibility of the model designs.

The reader is recommended to review the basic object-oriented terminology, concepts and symbolic notation that are presented in Appendix D before reading this chapter. Additionally, the Standard Template Library (STL), which is used to provide data storage functionality for the object design, is also discussed in Appendix D.

7.1 Object Model

The object model provides a structural point of view to describe the physical components and their relationships in the application. The main components of the object model are presented first and are followed by examples that demonstrate construction of the object model.

The problem of full field modeling has several subsystems that map directly into the object model. The high-level description of the physical system includes four subsystems: 1) *Field*, 2) *Reservoir*, 3) *WellGroupManager*, and 4) *Facilities*. The relationships among the subsystems are shown in Fig 7.1. The *WellGroupManager* system is used to store well group information and to perform collective operations on the wells. The *Reservoir*, *Well* and *Facilities* subsystems are further defined below.

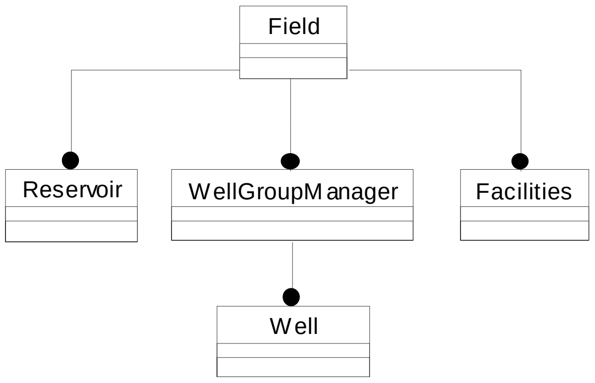


Figure 7.1: Basic Domains of the Object Model

Field

The field object has natural parallels with an actual field development scenario. A field can contain one or more reservoirs, wells, or facility models. The role of a *Field* object is to store these components and to coordinate their use with the objects and methods defined in the computational model. The components are stored in the container object, *list*, provided by the STL. Examples of *list* containers are presented in Appendix D.

Reservoir

The reservoir object also contains subsystems that parallel the physical description of a reservoir as well as numerical solution components required by the simulation formulation. The main subsystems of the reservoir object are shown and described below (Fig 7.2).

- The association between the *Reservoir* and *Rock* classes is one-to-one. The *Rock* class contains the compressibility, reference pressure data and methods to compute porosity for all the reservoir cells. The data is stored with respect to region number so that the reservoir model can be easily extended to incorporate multiple rock regions.

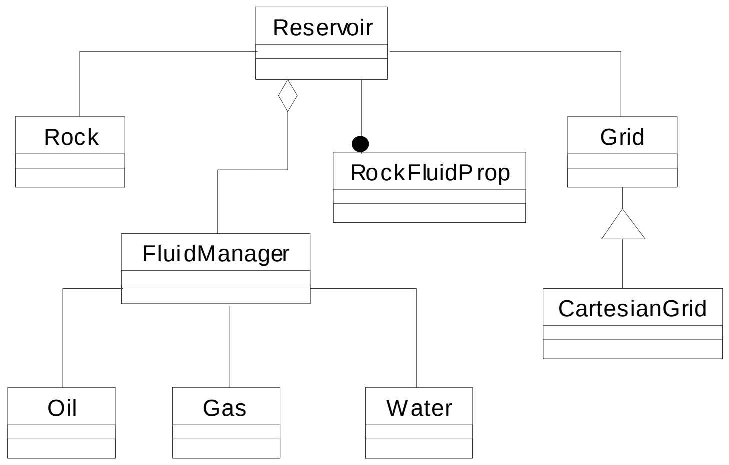


Figure 7.2: Reservoir Object Diagram

- A *FluidManager* object stores the fluid phases present in the reservoir. An aggregate relationship is assigned to the *Reservoir* and *FluidManager* classes since by definition, a reservoir contains fluids. There is a one to one association between the *FluidManager* and the *Oil*, *Gas*, and *Water* phase classes. The weak cohesion connected with an association relationship reflects the fact that not all phases may be present in the reservoir.
- The association between the *Reservoir* and *RockFluidProp* classes is one-to-many. The *RockFluidProp* is a base class with specializations for oil-gas and oil-water permeability properties (not shown). The specializations of the *RockFluidProp* class are discussed in context of the computational model presented in Section 7.3.
- The association between the *Reservoir* and *Grid* classes is one-to-one. The *Grid* class is a base class with a specialization for Cartesian grids. The *Grid* class provides a set of integer arrays for storing cell number mappings, number of connections per cell, and the connected cell numbers. Interface methods for accessing cell connectivity information are provided by the *CartesianGrid* class. The specialized *CartesianGrid* implementation provides interfaces for populating and accessing the data, thereby reusing the base class data storage methods.

Additionally the interface methods can be coupled with methods local to the specialized grid object for exploiting efficiencies associated with the grid layout.

The design of the grid classes demonstrates a fundamental difference in the designs between the code used hear and the codes of Verma (FLEX, 1996) and Nogaret (SPARTA, 1996). In FLEX and SPARTA, grid stencil information is stored in the form of connection objects, one object per connection. The connection object references cell objects that store cell property and state data. The number of cell objects equals the number of control volumes in the discretized model. This is in contrast to the design used here where a single *Grid* object stores all the connectivity information and cell properties in arrays accessed via data pool managers. The design of the data pool managers is discussed in Section 7.2.

Well

The *Well* object contains a list of completion and constraint objects and also has relationships with the facility model (Figure 7.3). As shown in Section 3.2.5, calculation of the total well rate requires individual completion rate data, which are functions of wellbore and well block properties. The completion object is responsible for computation of the well block flow rates and requires access to the well block information. An easy approach for accessing this information is by direct reference to the data vectors that store reservoir cell information. However, this results in a tight coupling between the *Reservoir* and *Well* objects and makes parallel well calculations difficult since a desired decomposition of well calculations may not correspond to the reservoir decomposition. The approach used here is to provide the completion object with a pointer to a copy of the global *Reservoir* object, which contains one cell corresponding to the relevant wellblock. The data required for well rate calculations is completely decoupled from the global reservoir object and facilitates parallel methods through compact data representation for message passing. An example that demonstrates reuse of the *Reservoir* class is presented Section 7.2 and is discussed with respect to construction of the preconditioning method.

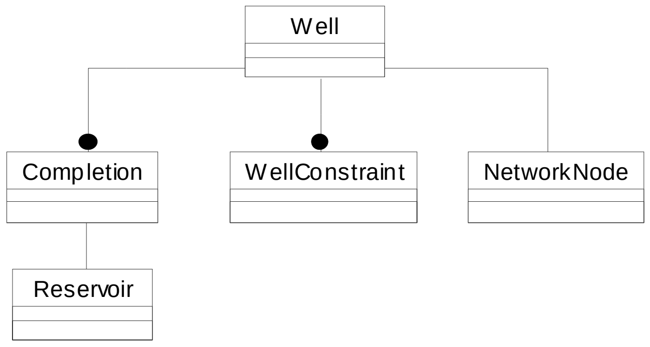


Figure 7.3: Well Object Model

Facilities

The object design for the facilities model was developed by Shaw and Byer (1998). The object hierarchy is shown in Figure 7.4. The *Facilities* object maintains a list of *NetworkNode* objects. Specializations of the *NetworkNode* class represent facility devices or connection points in the facility network. The base *NetworkNode* class provides interfaces for operations common to all nodes and the device specific implementations are provided via the specialized classes (*Junction, Pipe, Choke*). The base class also provides operations that are common to all types of nodes, such as

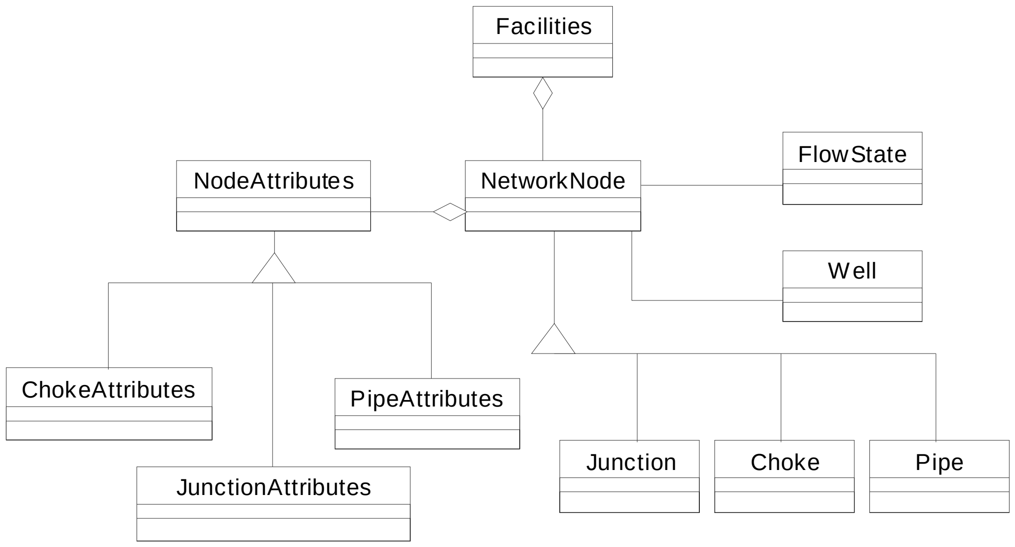


Figure 7.4: Facilities Object Model

constructing and maintaining a list of neighboring nodes. A similar relationship exists between the base class *NodeAttributes* and the specialized implementations. The *NetworkNode* class has an association with the *Well* class for obtaining reservoir inflow data.

7.2 Object Model - Construction

The key features in construction of the reservoir and facilities object model are presented and contrasted. Additionally, the data pool managers developed here are presented. The presentation also provides background information required for Section 7.3.

Object Model Instantiation

The reservoir and facilities model is defined using a keyword-based data input file. The keywords identify model data and are used internally the code to instantiate the object model. A *FileIO* object is employed to parse the data file via basic methods such as *popLine*, *popWord*, *popInteger,* and *popReal.* The *FileIO* object, *fio,* maintains a current file location pointer for the active object. The following code fragment (Fig. 7.5) illustrates how the *Field* object uses the *fio* object to instantiate *Reservoir* and *Facilities* objects:

```
Field::defineModel(FileIO & fio,DomMan *DM) {
    fio.popLine();                                          // read line from the data file
    fio.popWord();                                          // read first word from the line
    if( strcmp(fio.cv, "RESERVOIR") == 0 )  {               // is keyword RESERVOIR
        Reservoir *aReservoir = new Reservoir(fio,DM);         create Reservoir object and
        addReservoir(aReservoir);                              add object to STL container
    } else if( strcmp(fio.cv, " FACILITIES ") == 0 ) {      // is keyword FACILITIES
        Facilities *facilities = new Facilities (fio);         create Facilities  object and
        addFacilities(facilities);                            add object to STL container
    } else if …
```

Figure 7.5: Object Model Construction

Operations Required to Construct the Reservoir Object Model

A subset of the keywords and internal operations required for constructing the *Reservoir* object are shown in Table 7.1. The sequence of keywords and operations reflects the class hierarchy shown in Figs. 7.1 and 7.2. For example, in Table 7.1, the input line *GRID_DATA N50 CARTESIAN* indicates the beginning of a section of Cartesian grid data belonging to reservoir *N50.* Since the *Grid* object is associated with a *Reservoir* object, the *Reservoir* object is responsible for parsing the GRID input line. The next line of input *GRIDSIZE*, is related to the grid definition and therefore the *CartesianGrid* object is responsible for processing the data. The operation *addData* required by the next line of input, *DX*, introduces the data model employed.

Data Model

Since all reservoir cell properties can be classified by property name, size and unit type, a *DataVector* class is employed to abstract this information from the data storage methods. The class encapsulates the data and attributes, and provides access methods (Fig 7.6). The complete vector of property values can be accessed via the *Vec()* method and a similar method exists for accessing a single property value given a vector index.

The data pool scheme is demonstrated by examining the operations required for processing the *DX* keyword shown in Table 7.1. The *addData* method (Fig. 7.7) represents an interface to an operation that takes data, which can originate from sources other than file input, for example, intermediate computation results, and instantiates a *DataVector* object to encapsulate the data. The *DataVector* object is then inserted in STL *list* container object. An example of the method used to retrieve data from the STL *list* container via the data pool manager is shown in Appendix D.

By decoupling the property data and attributes from specific variable names, the program can accommodate the data requirements for new algorithms or formulations easily. Additionally, the data pool manager approach removes the memory requirements associated with storing a cell object for each control volume as in FLEX or SPARTA. It is interesting to note that the storage of data via vectors is commonly used

Table 7.1: Construction Process for Reservoir Object Model

Current File Location Pointer	Active Object	Operation Invoked
RESERVOIR N50 BLACKOIL OWG	Field	Reservoir *aRes = new Reservoir(fio,DM);
GRID_DATA N50 CARTESIAN	Reservoir	aGrid = new CartesianGrid(fio,DM);
GRIDSIZE 50 20 3	CartesianGrid	SetGridDimension(nx,ny,nz)
DX 500	CartesianGrid	addData("DX",x,nx,len,unittype);
KX 500	CartesianGrid	addData("KX",x,nxyz,len,unittype);
END_GRID_DATA	CartesianGrid	return to Reservoir Object
CELL_PROPERTIES N50	Reservoir	aGrid->readCellProp(fio,DM);
SWIR 0.12	CartesianGrid	addData(name,x,nxyz,unittype);
WSAT 0.12	CartesianGrid	SS.addData(name,x,nxyz,,unittype);
END_CELL_PROPERTIES	CartesianGrid	return to Reservoir Object
END_RESERVOIR	Reservoir	return to Field Object

```
class DataVector {
  // attributes
     char *name;                  // unique data id
     UnitType unittype;           // data unit type (SI or ENGLISH)
     int size;                    // number of data entries
     double *data;                // array storing data
  // operations
      DataVector(char *aname,
                 double *data,    // constructor method for loading
                 int size,           attributes
                 UnitType Aut);
     int Size();                  // return number of data entries
     double *Vec();               // return data vector
     double Vec(int offset);      // return single array value
}
```

Figure 7.6: Generic Data Storage Class - DataVector

```
void CartesianGrid::addData (char* name, double *data,     // generic addData method for
                             int n,UnitType ut)            // Cartesian grid data

     DataVector *dv = new DataVector(name,data,n,ut);      // create DataVector object

     listofData.insert(listofData.end(),dv);               // store data in listofData STL container
}
```

Figure 7.7: Example of Data Pool Manager for DataVector Objects

in Fortran programs, where data structures are limited to scalar and vector storage schemes.

A *SystemState* object, *SS*, stores time dependant properties of the grid using the same data pool manager methods discussed previously. Note the use of a similar *SS.addData* method for the *WSAT* keyword (Table 7.1). Also a *DomainManager* object, *DM* is shown in the argument list of several methods. This object provides information related to parallel and preconditioner configurations and is further discussed in Section 7.4.

Operations Required to Construct Facilities Devices

The method for constructing the *Facilities* object model is based on a design that decouples the operations required for assembling device attributes such as *PipeAttributes*,

Code Segment 1

```
Facilities::Facilities(FileIO &fio) {                    // constructs Facilities object from
                                                            input file specifications
    NetworkNode   *temp_node;                             // temporary pointers
    NodeAttributes *temp_attrib;
    fio.popWord();                                        // read first word of input line
    if( strcmp(fio.cva[0], "PIPE") == 0 ) {               // is it a PIPE specification
        temp_attrib = PipeAttributes::Create(fio);        // create attribute object that is
                                                            responsible for reading attributes
        temp_node=  Device::Create(Device::PIPE,          // send attribute type and data to Device
                                   temp_attrib);            factory that creates the node object
        AddDevice(temp_node);                             // insert node into STL list container
    }
}
```

Code Segment 2

```
NetworkNode * Device::Create (Device::Type type,
                              NodeAttributes &attrib_in) {  // create and return a specialized
                                                              NetworkNode object
switch (type) {
    case Facility::CHOKE:                                   // specialized Choke object created
        return Choke::Create(attrib_in);                      and attributes stored
    case Facility::PIPE:                                    // specialized Pipe object created
        return Pipe::Create(attrib_in);                       and attributes stored
```

Figure 7.8: Factory Method for Facility Devices

from the object to which they describe, thereby making the component classes for a facility device more specialized and limits the proliferation of unrelated methods in the same class. This is in contrast to the *CartesianGrid* class design, which requires local operations for computing grid connectivity information, in addition to the data input operations provided by the *Grid* object. The approach employed for the *Facilities* object is commonly referred to as the "*Factory*" method in computer science literature (Rumbaugh, et. al. 1991) due to the repetitive nature of construction. An example for the construction of a *Pipe* object is shown in Fig 7.8. Similar to the construction of the reservoir object, a *Facilities* constructor method (code segment 1) is used to control the instantiation of the Facilities object model. When a *PIPE* keyword is found, a *PipeAttributes* object is created that is responsible for reading the pipe attributes from the input file. The *Device* object is a *factory* used to create specific devices. The *Device* object's *Create* method (code segment 2) uses "type" information to defer (via the *switch* operation) object creation to the appropriate method. The same pattern of operations can be used for other devices and therefore increases the extendibility of the *Facilities* object model.

7.3 Computational Model

Coupled reservoir and facilities modeling is primarily a problem of simulation based on diversity of mathematical models, which are computationally intensive. The role of a computational object model is to capture algorithmic and performance requirements through operation specifications. Several views of the computational models used in this research are presented. The first view describes the operations and class structure required to simulate one timestep of a coupled model and defines a level abstraction common among other formulations provided here. The second view focuses on rock-fluid computations required within the solution of a timestep and is representative of the design employed for similar computations involving all reservoir cells. The final view represents an alternative design for coupling computational operations to the object structure and is based on the work by Shaw and Byer (1998).

Newton's Method

The primary operation required for simulation of a coupled reservoir and facility model is the solution of the nonlinear conservation equations over a timestep. As discussed in Section 3.3, Newton's method is employed for solving the nonlinear problem. The method requirements are used to define the computational model and form the basis for the class hierarchy and operations shown in Figure 7.9 and Table 7.2, respectively. The inputs to the *ResFacSolver* object include the following objects: 1) *Reservoir*, *WellGroupManager*, *Facilities* and *DomainManager*. These objects encapsulate all information related to their definition and current state, and represents a decoupling of the object model from the computational model. The operations provided by the computational model extract information from the object model using standardized interface methods. This allows for extensive reuse of the strategies and operations provided by the computational model. For example, the *ResFacSolver* class provides base methods for controlling the application of Newton's method to the explicit, standard, and preconditioned implicit formulations, each of which used the same Jacobian calculation operations. Through reuse of the control methods, combinations of the strategies were easily investigated and led to the preconditioned adaptive explicit formulation presented in Chapter 6.

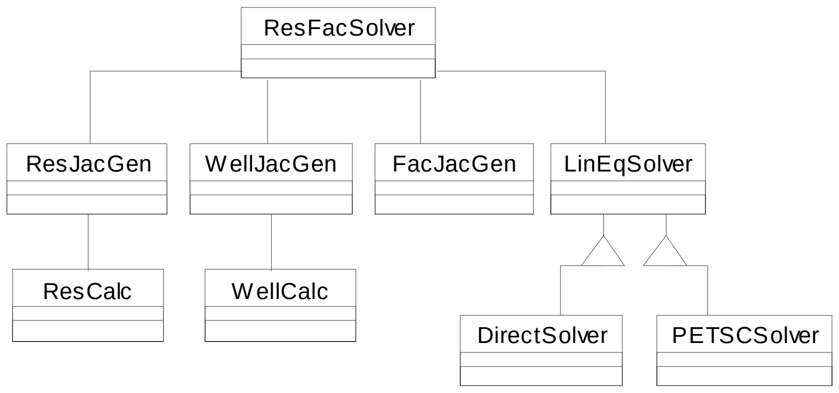


Figure 7.9: Computational Model Class Hierarchy - Numerical Solution Methods

Table 7.2: Computational Model Class Operations -- Numerical Solution

Class Name	Operation
ReservoirFacilitiesSolver	Assemble data and parameters required for the following Newton's method and the preconditioning technique.
ResCalc	Compute reservoir flow properties and/or Jacobian coefficients
ResJacGen	Provide interfaces for accessing ResCalc's methods for computing Jacobian matrix coefficients
WellCalc	Compute well rates and/or Jacobian coefficients
WellJacGen	Provide interfaces for utilizing WellCalc results
Facilities	Provide interface to methods for computing Facility Jacobian terms
LinEqSolver	Provide common interface methods for specialized solvers
DirectSolver	Provide interface methods for direct equation solver
PETSCSolver	Provide interface methods for iterative solvers

Additionally, by decoupling the computational components from the object model, the code can be easily extended to simulate development of a field containing several reservoirs that are brought on or taken off production at different times, and whose facility configurations may change with time.

Example of Reservoir Calculations

The computational model shown above describes classes designed to abstract specific calculations required to assemble Jacobian coefficients from the type of coupled formulation. An example of the design employed to compute specific components of the Jacobian coefficients is presented with respect to the operations required for relative permeability data. Specific code examples are restricted to the oil-water calculations. The data assembly operations and the structure of the calculations represent a design common to most of the computationally intensive operations found the code.

The *RockFluidProp* and *OilWatPerm* classes (Fig. 7.10 and Table 7.3) provide common interfaces for external access to the functionality provided by *OilWatPermImpl*, which is a specific implementation for computing oil-water relative permeability. A code sample is presented in Fig. 7.11 to illustrate how water relative permeability data is computed and incorporates several design features discussed previously.

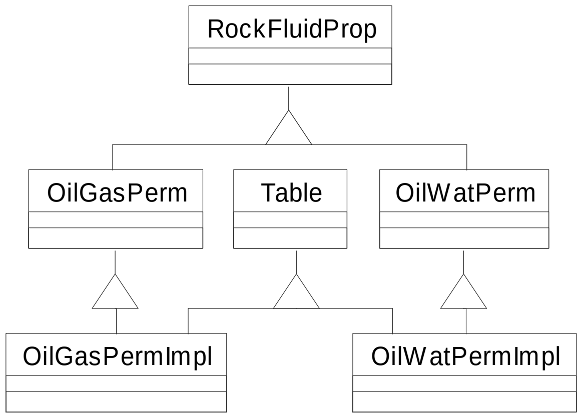


Figure 7.10: Class Hierarchy - Relative Permeability Computations

Table 7.3: Class Operations - Relative Permeability Computations

Class Name	Operation
RockFluidProp	Provide common interfaces for accessing rock-fluid property information
Table	Provide operations for table interpolation based on scalar or vector data input
OilWatPerm	Provide generic interfaces which abstract the computation of a oil-water permeability properties from the object requesting the property
OilWatPermImpl	Provide specific methods for computing oil-water permeability properties

Code Segment 1 - Method Requiring Water Relative Permeability Data

```
ResCalc::computeWatTrans(Reservoir *aRes)  {

    double *sw = aRes ->retGBProp("WSAT");        // access water saturation vector from data
                                                     pool manager

    RockFluidProp * OWRFP =                         // access RockFluidProp object corresponding to
            aRes->retRockFluidProp(OWPERM);            oil-water relative permeability properties

    int n = aRes->retGridIndex(NCELL);              // access number of reservoir cells

    double *krw = new double[n];                    // create storage space for n krw values

    OWRFP->krw(n,sw,krw);                           // retrieve krw's corresponding sw's for all cells
}
```

Code Segment 2 - Method Computing Water Relative Permeability

```
OWPermTableImpl::krw(int n,double *xsw,             // method for computing water relative
                        double *krw) {                 permeability via table properties
    sw = getData("SW")->Vec();                      // access water sat. vector from property table

    krw = getData("KRW")->Vec();                    // access krw vector  from property table

    Vinterpolate(n,xsw,xkrw,ntblentries,sw,krw);    // use Table's vector interpolation routine to
                                                    // compute krw's corresponding to xsw's
}
```

Figure 7.11: Example of Computational Objects for Relative Permeability

The *ResCalc* method accesses the required saturation and property information through interface routines provided by the *Reservoir* object (*aRes*), which subsequently accesses the requested information from data pool managers. The number of saturation values could have been obtained by the *DataVector* object that encapsulates the saturation data, however the information is obtained by interfaces to the grid description information to demonstrate how similar access methods are applied to grid connectivity data. The *OWPermTableImpl* operation employs similar data access methods for assembling the required property table components for input to an interpolation routine that operates on a vector of input values.

This example further illustrates the differences between the approach used here and that used by FLEX or SPARTA. In the process of computing Jacobian terms there are several operations, similar to relative permeability, that must be applied to all cells in the model. Here, the data are assembled for all the cells requiring a specific operation prior to execution of the operation, which is very similar to the Fortran style of programming. This is in contrast to FLEX or SPARTA, where calculation of Jacobian terms requires looping through all the cell or connection objects and within each object data are assembled and the operation is then executed. These differences can be classified broadly as code designs based on either an “array of objects” (FLEX and SPARTA) or an “object of arrays” (here).

Example of Facilities Calculations

The class design employed to provide the operations for Jacobian calculations in the facilities model is now presented. In contrast to the operations for the reservoir model where identical calculations are applied to a large amount of data, the Jacobian calculations required for the facilities model are a function of the device type, for example pipeflow versus choke calculations. The design employed is similar to the connection approach employed by FLEX and SPARTA, however the connectivity is stored via *list* containers provided by the STL versus connection objects that must be provided by the programmer as required in FLEX and SPARTA.

The class design and a sample implementation are shown in Figs 7.12 and 7.13. The *Facilities* class provides an interface method (code segment 1) for the timestep control routines that require Jacobian data for the facilities domain. The *Facilities* object creates a ProcessManager object that stores a *ProcessInfo* object. The *ProcessInfo* object encapsulates information that can be accessed by the objects performing the process operations (i.e. Jacobian calculations). The *Pipe* object provides an interface routine, *GenerateJacobian* for external access (code segment 2). Internally the *Pipe* object contains a pointer to a *PipeProcess* object that is responsible for all pipeflow related calculations. The *Pipe* object passes execution along with the *ProcessManager* object to the next connection. The next connection (device) is accessed via the same *GenerateJacobian* interface and represents how recursion is used in facilities related operations.

As shown in Fig. 7.12, computations are decoupled from the device object via the processing base. This follows the design for processing attribute information presented in Section 7.2. The class *ProcessManager* and *ProcessBase* methods do not fully exploit the level of operation decoupling possible using the process methods, which is evident by the methods whose names are process specific. Recommendations for improving the implementation are presented in Chapter 8. However the process design does contain

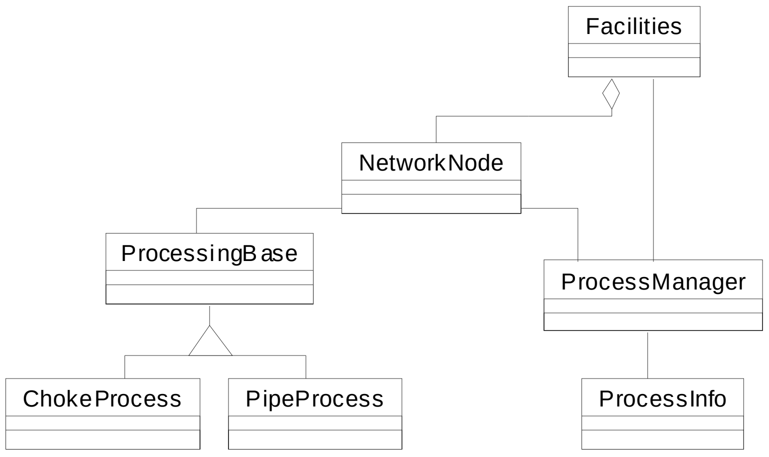


Figure 7.12: Computational Model - Facilities Classes

Code Segment 1

```
Facilities::GenerateJacobian() {                      // control method for Facilities Jacobian
                                                         calculations
    ProcessManager *pm =
                          new ProcessManager;         // create a ProcessManager object

    ProcessInfo *in; = new ProcessInfo;               // create a ProcessInfo object

    in->DebugOff();                                   // set process debug reporting flag in to off

    pm->ProcessInfoData(in);                          // store ProcessInfo object in ProcessManager

    NetworkNode     * root = returnStartNode();       // retrieve starting node of network tree
    root->GenerateJacobian(pm);                       // begin Jacobian calculations

}
```

Code Segment 2

```
Pipe::GenerateJacobian(ProcessManager *pm) {          // control method for device Jacobian calculations

    ProcessInfo *in = pm->ProcessInfoData();          // retrieve ProcessInfo object from
                                                         ProcessManager
    ProcessingBase * process =                        // create PipeProcess object for accessing
                             new PipeProcess;            computational methods

    process_ptr->GenerateJacobian(in);                // start Jacobian calculations for current pipe

    for(i= connlist.begin(); i != connlist.end(); ++i )
                                                      // recurs through Facilities nodes and compute
         (*i)->GenerateJacobian(pm);                     Jacobian terms

}
```

Figure 7.13: Example of Computational Objects for Pipe Device

several features that allow for significant reuse of the computational methods and provide pathways for directly extending the computations to parallel formulations, which are discussed further in Section 7.4.3.

7.4 Extendibility and Domain Oriented Reuse

Several components of the basic object design facilitated development of new solution methodologies and modeling capabilities. The three most notable examples are presented with respect to the preconditioning method and parallel formulation presented in Chapters 5 and 6, respectively. The first example demonstrates how *Reservoir* and

Grid objects are extended for construction of subdomains used by the Newton preconditioner. The second example demonstrates how the base class design is reused to construct reservoir domains for the parallel formulation. The final example demonstrates how the facilities class structure is reused and extended to provide parallelism in the facilities Jacobian calculations.

7.4.1 Construction of Preconditioner Subdomains

The preconditioning method requires construction of local reservoir domains. A *DomainManager* object stores subdomain configuration information and provides operations for constructing cell index mappings and defining domain boundary cells. The objects responsible for constructing subdomain objects can access this information using interface methods provided by the *DomainManager*. The primary operations required to construct subdomains are shown in Figure 7.14. Consistent with the design of other Newton control methods, the *ResFacSolver* object provides the method (code segment 1) that controls the construction and solution of the preconditioner.

The *Reservoir* and *CartesianGrid* object design is extended through addition of a specialized subdomain related constructor and copy methods (code segments 2 and 3). A reservoir subdomain is instantiated from a parent *Reservoir* object, *DomainManager* object and subdomain index. As noted above the *DomainManager* provides information that defines the subdomain configuration. This information is used by the new *CartesianGrid* object for copying grid and cell data. The specialized copy methods access the global data of the parent grid and then extract and store data required for the subdomain grid. Once the subdomain grid information is copied from the parent grid, the local grid connectivity is computed using existing class methods.

This example demonstrates how concepts in base object design are reused for operations required by the new formulation. The chain of operations for creating preconditioner subdomains follows directly from the operations used to create the parent reservoir from input data as shown in Table 7.1. Thus all that is needed to construct the subdomains are the addition of a few specialized constructor and copy

Code Segment 1 - Preconditioner Control Method

```
ResFacSolver::preconditionIteration( … ) {

       Int ndom = DM->retNDom ();                         // obtain number of subdomains

       Reservoir **SubDomReservoir =                      // create pointer array to reservoir objects
                       new * Reservoir[ndom]

       For(int domid=0; domid < ndom; domid ++)

         SubDomReservoir[domid] =                         //  construct reservoir subdomain from
                new  Reservoir(baseRes,DM, domid)         // base reservoir object

}
```

Code Segment 2 - Reservoir Class Method for Constructing Reservoir Subdomain from Parent Reservoir Object

```
Reservoir::Reservoir(Reservoir *baseRes ,                 // specialized constructor method for
                DomainManager *DM,                        // creating a local reservoir object from a
                int domid ) {                             // base reservoir object

     DM->loadSubdomainIJK_Bndry(&i1,&i2,                  // retrieve IJK indices that define
                        &j1,&j2,                          //  subdomain boundaries
                        &k1,&k2)

    aGrid =  new CartesianGrid(DM,domid,                  // create subdomain grid from base
                      baseRes ->aGrid,                    // reservoir's grid
                      i1,i2,j1,j2,k1,k2);

}
```

Code Segment 3 - CartesianGrid Class Method for Constructing a Local Grid from Parent Grid Object

```
CartesianGrid::CartesianGrid(DomMan *DM,int domid,
                      Grid *aGrid,                        // specialized constructor method for
                       int i1,int i2,                     // creating a local Cartesian grid object
                       int j1,int j2,                     // from a base grid object
                       int k1,int k2):Grid(aGrid)  {

     copyData(DM,aGrid,i1,i2,j1,j2,k1,k2);                // copy local grid and cell property data

     SS.copyVars(DM,                                      // retrieve base grid's reservoir state
                aGrid-retReservoirState(),                // object and copy state for local grid
                i1,i2,j1,j2,k1,k2,);

                                                          // construct connectivity information based
      setup7PT_GS();                                      //  from local grid definition

}
```

Figure 7.14: Extendibility and Domain Reuse - Preconditioner Construction

methods, the data pool manager and computational methods are completely reused. Additionally, since the subdomain objects are defined using native *Reservoir* class definitions, the Jacobian generation operations supplied in the computational model are reused in solution of the preconditioner. The following example shows how the parallel design further leveraged the base object design.

7.4.2 Construction of Reservoir Domains for Parallel Formulation

The construction of reservoir domains for the parallel formulation requires extending the object design to provide operations for data sharing and parallel data input. The parallel formulation requires that copies of the same program execute on different processors and each program is responsible for only one reservoir partition. Therefore the methods for construction of a reservoir object in a sequential formulation can be reused in the parallel formulation with only minor, highly localized modifications. Similarly many of the *DomainManager* class methods developed for cell index mapping operations required by the preconditioning method are reused to provide mapping operations for the reservoir decomposition.

The sequential and parallel formulations use the same input file for building the object model, however in the parallel formulation one processor is designated as the master node and is responsible for reading and broadcasting the input file data. Recall that the construction process for the sequential formulation utilized the *getLine*() operation for parsing the input file, which was provided by the *FileIO* object. The *getLine*() method is modified to provide parallel data input and enables significant reuse of the object model class structure. The objects that use the *getLine*() operation also use *DomainManager* operations to identify partitions of the broadcast data. The example shown in Fig. 7.15 demonstrates the operations for parallel processing of cell property data. In this example assume that all the processes are synchronized at the *popLine()* operation in the method *readCellProp()* (code segment 1). The processes enter the *popLine()* method (code segment 2) and the master process reads from the input file and

Code Segment 1 - Object Requiring FileIO Methods

```
CartesianGrid::readCellProp(FileIO &fio,
                            DomMan *DM) {
      fio.PopLine();                          // read the next line of input data
      if ( checkValidKW(fio.popWord() ) {     // check for valid keyword
         loadArray(fio,x,nxyz,DM);            // load property array from FileIO
                                              // object into local data array
         addData(aname,x,nxyz,,comp_unit);    // add data to data pool manager
     }
}
```

Code Segement 2 - FileIO popLine Method - Sequential and Parallel Version

```
FileIO::popLine() {
      if (myid == 0 && ndomains > 1 ) {       // check if master process and parallel run
         i = SeqPopLine();                    // read data for global domain and store
                                              // character data array labeled line
          for(int to=1;to<ndomains;to++)      //  send line of input to slave processes
              MPI_Send(line, length, MPI_CHAR,to,  // via MPI supplied functions
                    tag, MPI_COMM_WORLD);
      } else if  ( ndomains > 1 ) {           // check if parallel run and slave process
         MPI_Recv(line, count, MPI_CHAR, from,
                 tag,MPI_COMM_WORLD,          // receive data for global domain via MPI
                 &status);                    // supplied functions
      } else                                  // sequential run
         SeqPopLine();                        // interface routine to low level C++ input
                                              // methods
}
```

Code Segment 3 – loadArray method - Sequential & Parallel Version

```
CartesianGrid::loadArray(FileIO &fio,
                         double *x,           // copy data from FileIO object into local
                         int nentry,             array
                         DomainManager *DM) {
        for(j=0;j<fio.naval;j++)
          tempx[j] = fio.PopReal();           // pop data from line
        if (DM->ndomains > 1)                 // if parallel run use information provided by
                                              // DomainManager to extract local domain
           copyDataWindow(x, tempx,DM)        // data from tempx containing global data
        else                                  // sequential run, extract all data from
           copyData(x, tempx)                 // tempx
}
```

Figure 7.15: Parallel Input and Object Model Construction Methods

sends the data to the slave processes using MPI send functions. The slave process bypasses the file read operations (*SeqPopLine()*) and receive the data using MPI receive functions. Each processor now has a copy of the file-input line, which represents data for the global problem and does not contain any information related to data decomposition. The *loadArray*() method is employed to copy the data from the *FileIO* object into temporary arrays. Within the *loadArray*() method (code segment 3) the *copyDataWindow*() method is employed, which uses domain information supplied by the *DomainManager* to copy the local reservoir domain data from the global data vector.

The basic design employed for parallel processing of cell property data is reused throughout the object model. The *FileIO* and *DomainManager* objects work together to provide data sharing and domain partitioning information for the object construction methods. This example demonstrates the ease with which the sequential code is extended to a parallel formulation. The modifications to the object model are very localized and also allow for extensive reuse of methods provided by the computational model.

7.4.3 Parallel Computation of Facilities Jacobian Coefficients

The object and computation model design developed for the sequential formulation was almost entirely reused in the parallel formulation. The *ProcessManager*, *ProcessInfo*, and *ProcessingBase* classes provided sufficient decoupling of device computations from the formulation so that the parallelization of the Jacobian generation process could be abstracted at a high level in the facility model class structure, thereby removing any issues associated with formulation from the lower level class methods. No new objects are required, only the *ProcessManager* and specialized *NetworkNode* classes required new methods. The *ProcessManager* is the control routine for Jacobian calculations and provides methods for both the manager-worker and worker-worker parallel formulations discussed in Section 6.1.3.

A pseudocode example that shows the key methods employed to facilitate discussion of the class modifications and extensions (Fig. 7.16). The *ProcessManager* uses the *Facilities* and *DomainManager* objects to obtain respective facilities and

Code Segment 1

```
ProcessManager::GenerateJacobian() {
   if  (DM->ManagerCPU() ) {                              // if manager setup task list
     SetupTaskList();                                     // setup prioritized task list
     ProcessTask(GENJAC);                                 // process tasks
     RegisterProcess(RETURN);                             // computation finished broadcast RETURN
   } else {
        WaitForProcess();                                 // worker  - wait for assignments
   }
}
```

Code Segment 2

```
ProcessManager::ProcessTasks(TaskType tasktype) {        // control method for task assignment
   while ( SourceAvailable() )                            // process parallels tasks starting with
                                                          SOURCE node
      if ( CPU_Available() )  {                           //  is there a CPU available
         RegisterProcess(NextCPU(),                       //  assign the next CPU available the task of
                         NextSource(),                    Jacobian generation to the next SOURCE
                         tasktype);                       node available
      } else if (ManagerDoesWork  ) {                     // no CPU's available
         RegisterProcess(DM->ManagerCPU(),
                         NextSource(),                    // if manager does work assign it a task
                         tasktype);
      } else {
         WaitForProcess();                                // wait for worker nodes to finish task
      }
}
```

Code Segment 3

```
ProcessManager:: WaitForProcess () {                      // worker control method
x0:   message = ReceiveMessage();                         //  wait for message
      switch (message) {
         case RETURN;                                     //  return, no more assignments
            return;
         case GENJAC;                                     // Jacobian calculation task
                                                          // receive task assignment details,
            i = ReceiveProcessInfo();                        current node state and encapsulate in
                                                             ProcessManager object
           Facilities->node[i]-GenerateJacobian(pm)       // start computations
           SendResults();                                 // send results back to manager
      }
goto x0;                                                  // wait for more messages
```

Figure 7.16: Process Management for Parallel Facilities Jacobian Calculations

solution domain information, such as the list of *NetworkNode* objects that require computation and the number of available processors. The master process (code segment 1) constructs a task list using the method *SetupTaskList()*, which also initializes processor and task queues used by the *ProcessTask()* method. The *ProcessTask()* method (code segment 2) embodies the parallel computation strategy discussed in Section 6.1.3. The source nodes form the first set of parallel tasks as indicated by the *SourceAvailable()* method. A similar execution loop (not shown) exists for the next level of parallel computation defined by *Junction* nodes. The *RegisterProcess* method is used by the manager node to send task assignment and device state data to a worker node, which resides in a wait state until the messages and data are received. The worker node receives and encapsulates the process information in a *ProcessInfo* object (code segment 3). The *FlowState* object (Fig. 7.4) provides methods that encapsulate MPI functions for sending and receiving state data. Once all the state data is received, the Jacobian computations are started using the same methods employed by the sequential formulations. The specific device method *GenerateJacobian* is modified to use *ProcessInfo* data from the *ProcessManager* object to determine when to stop computation. This check is required due to the recursive nature of the network calculations and therefore the stop calculation signal must come from data available with the recursive flow calculation.

The control and execution processes described here demonstrate the reusability of the facilities class structure. Only minor highly localized modifications to the specialized implementations of the *NetworkNode* class are required. The extensions required in order to achieve the data passing functionality are also highly localized and reside in specialized methods that have no impact on the sequential formulation. Additionally, the techniques for achieving parallelism (manager-worker, worker-worker) are restricted to the *SetupTaskList()* and *ProcessTask()* methods. The *RegisterProcess()* method, which is responsible for the critical support methods of assigning tasks and sending/receiving results, is decoupled from the specific parallel formulation. Therefore, the majority of the modifications required for testing a new formulation are with respect to the code implementation for the strategy and the support methods can be reused, which enables the code developer to focus primarily on algorithmic issues and less on coding requirements.

7.5 Concluding Remarks

The major features of the object-oriented design of the research code have been presented. Several examples serve to demonstrate the reusability of the object model data structure and computational methods. Additionally, the object and computational model design provide significant insights into the design of objects and methods required for the new preconditioning technique and parallel formulations.

Chapter 8

Conclusions and Recommendations

8.1 Conclusions

The primary focus of this research is to develop methods for increasing the computational efficiency of fully coupled implicit reservoir and surface facility problems, that can utilize both sequential and parallel processing environments. This research also focuses on the object-oriented design of the application code with the goals of reusability and extendibility. The following comments and conclusions can be made as a result of this research:

Surface Facilities Models

1. Test results for surface facility problems with specified source and sink boundary conditions indicate that the number of Newton iterations required for convergence can be significantly higher than the number of iterations required for a reservoir problem with specified well boundary conditions.
2. The solution procedure must be capable of resolving flow reversals associated with a poor initial solution or equipment reconfigurations. Newton step scaling may improve the convergence rate, however, the scaling factor is problem dependent.

3. Flow reversal in the pipe network can be resolved using Newton's method. The residual terms can be modified to force the method to generate solution updates that serve to lower or raise a node pressure until flow can occur. This technique represents a simplified approach improving the robustness of Newton's method.

Coupled Reservoir and Surface Facilities Models

4. The explicit formulation decouples the reservoir and facility domains and allows for efficient solution of the nonlinear reservoir problem at the expense of material balance errors at the reservoir and facilities domain interfaces.
5. The standard implicit facilities formulation eliminates the material balance errors to a specified tolerance. However the convergence rate is similar to that observed when solving only the facility domain. Associated with each additional Newton iteration is the cost of the linear solve using a Jacobian matrix containing coefficients for the entire reservoir flow field, which considerably increases the CPU requirements.
6. The adaptive implicit reservoir formulation reduces the cost of solving for the reservoir flow field, but similar to the standard implicit facilities formulation, the full facilities coupling results in increased CPU requirements
7. An adaptive explicit facilities formulation has been developed which combines the efficiency of the explicit formulation with accuracy close to that of the standard implicit formulation. The method is based on the observation that the most significant material balance errors associated with the explicit formulation occur during transition periods defined by the occurrence of two-phase flow in the wellbore. The method contains criteria

based on wellbore phase behavior for switching between explicit and implicit formulations.

8. A new preconditioning method has been developed that can be used with the standard implicit and adaptive explicit facilities formulation to accelerate convergence of the fully coupled iterations. Test results show that the computational cost of the standard implicit formulation can be reduced by 15 to 40%. Additionally, the preconditioned adaptive explicit formulation is capable of solving the fully coupled reservoir and facilities problem within an acceptable level of material balance error while reducing the CPU requirements to that of the explicit facilities formulation.
9. The preconditioning method can also be used with an adaptive implicit treatment of the reservoir flow field. The preconditioner well subdomains are chosen to coincide to the initial boundaries for implicitness, resulting in a preconditioning of the fully coupled adaptive implicit reservoir problem.
10. The preconditioned standard implicit facilities formulation was developed to run in sequential and parallel computing environments. The parallel linear solve is the most expensive component in the standard implicit formulation. In the preconditioning method, the facilities Jacobian calculations were found to be the most expensive component.
11. Two strategies for achieving parallel facilities Jacobian calculations were investigated. The worker-worker model distributed the computational tasks among all processors. One of the processors was designated as the manager and also performed task management operations. The manager's strategy for balancing task assignment with local computations severely restricted parallel efficiency beyond two processors. The manager-worker model was designed to maximize the computation efficiency of the worker processors at

the expense of limiting the manager's responsibility to task management. Therefore the parallel model restricted the overall parallel efficiency. The scalability for both of these methods is limited and suggestions for improving efficiency are presented.

12. The major features of the object-oriented design of the code have been presented. The decoupling of the object and computation model components of the design is critical to achieving the features of reusability and extendibility. The object-oriented approach allows for rapid development of methods for solving coupled systems. The object design applied data encapsulation techniques to provide consistency with the physical description of the model and to provide a strong cohesion of the object entities. The weak coupling of the computational and object models enhances the reusability and extendibility of the numerical methods. Additionally a parallel object model was easily obtainable with only minor highly localized modifications to the object construction methods.

8.2 Recommendations for Further Study

The research presented in this dissertation has addressed the important issues of efficiency and accuracy of methods for solving coupled reservoir and surface facilities models. The following recommendations pertain to simulator development items and to research areas for improved coupled reservoir and facility modeling.

1. The surface facilities model employs a simplified homogeneous model for pipeflow. This model could be replaced with advanced formulations that are capable of modeling the flow regimes commonly present in pipeflow.
2. The surface facility model allows for pipe, choke, and simple separator devices. The model could be extended to include compressor, pump, and

rigorous separation devices to allow for more comprehensive coupled model configurations. When appropriate the device attribute should be allowed as a primary variable in the formulation. For example a solution variable for the compressor device may be the horsepower requirements.

3. The pipe network is limited to gathering systems with tree-like structures. The network and solution procedure should allow for looped networks as a method for increasing production capacity.
4. Newton's method with simple scaling and heuristic rules was employed to solve the nonlinear problem. Other methods for solving the nonlinear problem should be investigated with the objectives of improving efficiency and robustness of the solution procedure
5. A formulation, which combines the adaptive reservoir flow field and adaptive explicit facility coupling formulations, will capitalize on the individual efficiencies of each method and should be developed. The efficiency of the facility coupling could be further improved through the preconditioning method, resulting in a preconditioned adaptive implicit reservoir adaptive explicit facilities formulation.
6. The linear equation solver for both the sequential and parallel formulations restricted the size of the reservoir model. More efficient methods for solving linear systems are essential for the modeling of coupled systems that contain more realistic reservoir descriptions, thereby allowing for more complex flow patterns that can complicate the design and operation of the surface facilities model. Additionally, the impact on the facility domain equations on current preconditioning methods used by the linear solver should be investigated.

7. Parallelization of the facilities Jacobian calculations should be further investigated. The current manager-worker task allocation scheme should be redesigned so that the processor utilization is maximized for arbitrary numbers of processors. This would require incorporating cost per task information, which is obtained via previous iteration performance, into the allocation scheme.
8. Optimization based algorithms should be investigated that allow for improved evaluation of alternate development scenarios. While global optimization is currently impossible, methods designed to improve the decision making process over a limited production horizon can employ optimization methods. For example, given a set of specified objectives coupled with alternate action plans, if those objectives are not achieved, an optimization algorithm can be employed to evaluate the best action alternative when the objectives are not met. The challenge associated with applying optimization techniques to coupled models is related to the evaluation of the objective function, which requires solution of the nonlinear problem. However, the preconditioning method also provides a basis for considering optimization methods. The preconditioning method is defined by a local problem, which is easy to solve and can be employed as a coarsened representation of the full model description, and thereby provides for more efficient iterations in the optimization algorithm.
9. The object-oriented design developed for the facilities model can be further improved by abstracting the Jacobian generation process from the *ProcessManager* object, thereby simplifying the extensions of the parallel capability provided by the *ProcessManager* to other calculations required for the surface facilities, such as solution estimation or optimization.

Nomenclature

A	pipe cross-sectional area or coefficient matrix
A_p	area occupied by phase p
A_x	grid block area in the x-direction
$\hat{A}$	Schur complement or capacitance matrix
A_{cf}	matrix of Jacobian coefficients defining coarse to fine grid coupling
A_{fc}	matrix of Jacobian coefficients defining fine to coarse grid coupling
A_{sp}	matrix of Jacobian coefficients for preconditioner subproblem
A_f	matrix of Jacobian coefficients for fine grid or facilities domain
A_c	matrix of Jacobian coefficients for coarse grid
A_g	matrix of Jacobian coefficients for reservoir and facility domain
A_r	matrix of Jacobian coefficients for reservoir domain cells
A_w	matrix of Jacobian coefficients for well equation coefficients
A_{rw}	matrix of Jacobian coefficients defining reservoir to well coupling
A_{wf}	matrix of Jacobian coefficients defining well to facilities coupling
A_{wr}	matrix of Jacobian coefficients defining well to reservoir coupling
A_{fw}	matrix of Jacobian coefficients defining facilities to well
$A^{h'}$	matrix of Jacobian coefficients for grid level h'
B_p	formation volume factor of phase p
C_d	distribution coefficient
d	pipe or tubing diameter
d_{NI}	difference in preconditioned standard and standard implicit Newton iterations
D	vertical depth
e_p	parallel efficiency
$e_{p,a}$	absolute parallel efficiency

$e_{p,r}$	relative parallel efficiency
E_p	phase holdup
f	friction factor
g	acceleration due to gravity
h	fine grid
h'	coarse grid
$I_h^{h'}$	linear restriction operator from the fine to coarse grid
$I_{h'}^{h}$	piecewise constant prolongation operator from the coarse to fine grid
J	Jacobian operator
J_r	Jacobian matrix for reservoir coefficients
J_f	Jacobian matrix for facilities coefficients
k	diagonal permeability tensor
k_{rp}	relative permeability of phase *p*
$\tilde{M}_{cp}$	mass injection/production rate of component *c* in phase *p* per unit reservoir volume
M_{cp}	mass injection/production rate of component *c* in phase *p*
n_c	number of components
n_{cpu}	number of processors
n_p	number of phases
n_{pv}	number of primary variables
p	pressure
p_{bp}	bubble-point pressure
p_o	oil phase pressure
p_w	water phase pressure
p_g	gas phase pressure
p_{sep}	separator pressure
p_{wb}	wellbore pressure
P_{cow}	oil-water capillary pressure
P_{cgo}	gas-oil capillary pressure

p_{rb}	reservoir boundary pressure
q_c	component mass flow rate
q_{sp}	specified mass flow rate
r	pipe or tubing radius
r_o	equivalent wellbore radius
r_w	wellbore radius
r_{well}	well equation residual
R	residual operator
$R_{c,l}$	mass balance error for component *c*, cell *l*
R_e	Reynolds number
R_{cp}	solubility of component *c* in phase *p*
s	skin factor
S	pipe perimeter or surface area
S_p	saturation of phase *p*
$T_{cp,l}$	transmissibility coefficient of component *c* in phase *p* in the *l*-direction
$\tilde{T}_{cp,l}$	transmissibility of component *c* in phase *p* in the *l*-direction
$T_{g,l}$	geometric factor in the *l*-direction
T_{seq}	CPU time required for sequential run
$T_{n_{cpu}}$	CPU time required for n_{cpu} processor run
t	time
$t_{j,o}$	the total time spent by processor *j* on an operation *o*
$\overline{t}_o$	the average time spent on an operation when considering all processors
$u_{j,o}$	processor *j* utilization for an operation *o*
U_b	bubble rise velocity
U_p	phase velocity
U_m	mixture velocity
U_{so}	oil phase superficial velocity
U_{sg}	gas phase superficial velocity
V	volume

V_b	bulk grid block volume
$\vec{n}$	unit outward normal vector
s	surface area
$R_{c,l}$	mass balance residual for component c , cell l
$\vec{R}_w$	residual vector for well equations
$\vec{R}_r$	residual vector for reservoir flow equations
$\vec{R}_f$	residual vector for fine grid reservoir flow equations or facility domain
$\vec{R}_c$	residual vector for coarse grid reservoir flow equations
$\vec{R}_g$	residual vector for reservoir and surface facility flow equations
$\vec{R}_{sp}$	residual vector for preconditioner subproblem
$\vec{v}_c$	component velocity vector
$\vec{v}_p$	phase velocity vector
WI	well index
x	primary variable
x_{cp}	mass fraction of component c in phase p
$\overline{X}_{pv}$	set of primary variables
$\vec{X}_r$	vector of unknowns for reservoir flow equations
$\vec{X}_w$	vector of unknowns for well equations
$\vec{X}_f$	vector of unknowns, fine grid reservoir flow equations or facility domain
$\vec{X}_c$	vector of unknowns for coarse grid reservoir flow equations
$\vec{X}_g$	vector of unknowns for reservoir and surface facility flow equations
$\vec{X}_{sd,i}$	vector of unknowns for well subdomain i
$\vec{X}_{sp}$	vector of unknowns for preconditioner problem
Δt	time interval
Δx	grid block length in x-direction
Δy	grid block length in y-direction
Δz	grid block length in z-direction

Greek Letters

β_p	coefficient that indicates if rate of phase *p* is specified
γ_l	wellbore gravity for completed layer *l*
η	maximum fraction change in previous solution value
τ	material balance error relaxation factor
τ_{wm}	mixture-wall friction shear stress
$\varsigma_{\max}$	threshold parameter for switching from explicit to implicit facilities formulation
ς	percent change in solution gas-oil ratio over consecutive timesteps
ε_R	maximum material balance error
ρ_c	component density
ρ_c^*	component density at standard conditions
ρ_m	mixture density
ρ_p	phase density
μ_m	mixture viscosity
μ_p	viscosity of phase *p*
υ	current Newton iteration number
υ_{fp}	first preconditioned Newton iteration of timestep
υ_{lp}	last preconditioned Newton iteration of timestep
υ_{lnp}	last nonpreconditioned Newton iteration of timestep
υ_{pt}	preconditioning threshold
υ_{sp}	first Newton iteration that preconditioning can be considered
υ_t	total number of Newton iterations required for a timestep
$\upsilon_{t,PcSI}$	total number of preconditioned standard implicit Newton iterations required for a timestep
$\upsilon_{t,SI}$	total number of standard implicit Newton iterations required for a timestep
ϕ	reservoir porosity
σ	interfacial tension
∇	gradient

Δ_t	time difference operator
Φ_p	fluid potential of phase *p*
Ψ	time dependant property
Ω_l	set of all grid blocks connected to block *l* or set of all nodes connected to node *l*

Subscripts

c	component
g	gas phase
o	oil phase
m	mixture
i, j, k	counter in summation terms

Superscripts

n	timestep level
υ	Newton iteration level

Abbreviations

CPU	central processing unit
SI	standard implicit facilities coupling formulation
PcSI	preconditioned standard implicit facilities coupling formulation
PcAI	preconditioned adaptive implicit facilities coupling formulation
PcAE	preconditioned adaptive explicit facilities coupling formulation
Exp-Fac	explicit facilities formulation
SI-Fac	standard implicit facilities formulation
AE-Fac	adaptive explicit facilities formulation
PcSI-Fac	preconditioned standard implicit facilities coupling formulation
PcAE-Fac	preconditioned adaptive explicit facilities coupling formulation

Bibliography

[1] Ames, W.F.: *Numerical Methods for Partial Differential Equations,* Academic Press, New York City (1977), second edition.

[2] Arnold, K., Stewart, M.: *Surface Production Operations, Design of Oil Handling Systems and Facilities*, Gulf Publishing (1986), Volume 1.

[3] Aziz, K.: Notes for Petroleum Reservoir, Stanford University (1995).

[4] Aziz, K., and Settari, A.: *Petroleum Reservoir Simulation*, Applied Science Publishers ltd., London (1979).

[5] Balay, S., Gropp, W.D., McInnes, C., and Smith, B.F.: "Efficient Management of Parallelism in Object Oriented Numerical Software Libraries," *Modern Software Tools in Scientific Computing*, E. Arge, A.M. Bruaset, and H.P. Langtangen, eds. Boston, Massachusetts: Birkhauser Press (1997).

[6] Balay, S., Gropp, W.D., McInnes, C., and Smith, B.F.: "PETSc home page", "*http://www.mcs.anl.gov/petsc*," (1999).

[7] Balay, S., Gropp, W.D., McInnes, C., and Smith, B.F.: "PETSc 2.0 Users Manual", ANL-95/11 - Revision 2.0.24, Argonne National Laboratory (1999).

[8] Beggs, H.D.: *Production Optimization*, OGCI Publications, Oil and Gas Consultants International Inc., Tulsa (1991).

[9] Bell, J.B., Trangenstein, J.A., and Shubin, G.R.: “Conservation Laws of Mixed Type Describing Three-Phase Flow in Porous Media,” *SIAM J. Appl. Math*. (Dec. 1986), 1000-1017.

[10] Booch, G.: *Object-Oriented Analysis and Design with Applications*, The Benjamin/Cummings Publishing Company, Inc. (1994).

[11] Bramble, J.H., Ewing, R.E., Pasciak, J.E. and Schatz, A.H.: "A Preconditioning Technique for the Efficient Solution of Problems with Local Grid Refinement," *Comput. Meth. Appl. Mech. Eng.* (1988), 149-159.

[12] Briggs, W.L.: *A Multigrid Tutorial*, Lancaster Press, Pennsylvania (1987).

[13] Brill, J.P. and Beggs, H.D.: *Two-Phase Flow in Pipes*, Tulsa (Feb. 1991).

[14] Brown, K.E. and Lea, J.F.: “Nodal Systems Analysis of Oil and Gas Wells”, SPE Distinguished Author Series (Oct. 1985).

[15] Byer, T.J., Edwards, M.G., and Aziz, K.: “Preconditioned Newton Methods for Fully Coupled Reservoir and Surface Facility Models,” SPE paper 49001 (1998).

[16] Byer, T.J., Edwards, M.G., and Aziz, K.: “A Preconditioned Adaptive Implicit Method for Reservoirs with Surface Facilities,” SPE paper 51895 (1999).

[17] Chien, M.C.H., and Northrup, E.J.: “Vectorization and Parallel Processing of Local Grid Refinement and Adaptive Implicit Schemes in a General Purpose Reservoir Simulation” SPE paper 25258 (Feb. 1993), 279-290.

[18] Chien, M.C.H., Tchelepi, H.A., Yardumian, H.E., and Chen, W.H.: “A Scalable Parallel Multipurpose Reservoir Simulator,” SPE (June 1997), 17-30.

[19] Deimbacher, F.X., Komlosi, F., and Heinemann, Z.E.: ”Fundamental Concepts and Potential Applications of the Windowing Technique in Reservoir Simulation”, SPE paper 29851 (Mar 1995), 105-117.

[20] Edwards, M.G., and Rogers, C.F.: “A Flux Continuous Scheme for the Full Tensor Pressure Equation,” proceedings, 4th European Conference on the Mathematics of Oil Recovery, Roros (1994).

[21] Emanuel, A. S., and Ranney, J.C.: “Studies of Offshore Reservoir with an Interface Reservoir/Piping Network Simulator,” *JPT* (Mar. 1981), 399-406.

[22] Forsyth, P.A. and Sammon, P.H.: “Practical Considerations for Adaptive Implicit Methods in Reservoir Simulations,” *J. Comp. Phys.* (Feb. 1986) 62, 265-81.

[23] Forsyth, P.A. and Sammon, P.H.: “Quadratic Convergence for Cell-Centered Grids,” *Appl. Numer. Math* (1988) 4, 377.

[24] Foster, I.: *Designing and Building Parallel Programs*, Addison-Wesley Publishing Company (1994).

[25] Fujii, H.: "Multivariate Production Systems Optimization in Pipeline Networks", M.S. Thesis, Stanford University, August 1993.

[26] Fung, L.S., Collins, D.A. and Nghiem, L.X.: “An Adaptive-Implicit Switching Criterion Based on Numerical Stability Analysis,” *SPER* (Feb. 1989), 45-51.

[27] Golub, G.H. and Ortega, J.M.*: Scientific Computing: An Introduction with Parallel Computing*, Academic Press, Boston (1993)

[28] Govier, G. W. and Aziz, K.: *The Flow of Complex Mixtures in Pipes*, Van Nostrand Reinhold Co., New York City (1972).

[29] Grabenstetter, J., Li, Y.K. and Nghiem, L.X.: “Stability-Based Switching Criterion for Adaptive-Implicit Compositional Reservoir Simulation,” SPE (Feb. 1991), 243-258.

[30] Gropp, W., Lusk, E., and Skjellum, A.: *Using MPI, Portable Parallel Programming with the Message Passing Interface*, The MIT Press (1994).

[31] Hepguler, G., Barua, S., and Bard, W.: “Integration of a Field Surface and Production Network with a Reservoir Simulator,” SPE paper 38937 (Apr. 1997).

[32] Jain, A.K.: "Accurate Explicit Equation for Friction Factor", *ASCE Hydraulics Div. J.* (1976), 102(HY5): 674-677.

[33] Jeppson, R.: *Analysis of Flow in Pipe Networks*, Ann Arbor Science Publishers, Inc., Michigan (1977).

[34] Killough, J.E., and Wheeler, M.F.: “Parallel Iterative Linear Equation Solvers: An Investigation of Domain Decomposition Algorithms for Reservoir Simulation,” SPE paper 16021 (Feb. 1987) ,294-312.

[35] Lim, K.T., Schiozer, D.J., and Aziz, K.: "A New Approach for Residual and Jacobian Array Construction in Reservoir Simulators," *SPE Computer Applications* (August 1995), No. 4, 93-97.

[36] Lippman, S.B.: *C++ Primer*, Addison-Wesley Publishing Company (1989).

[37] Litvack, M., Clark, B., Farichild, J., Fossum, M., Macdonald, C., Wood, A.: “Integration of Prudhoe Bay Surface Pipeline Network and Full Field Reservoir Models”, SPE (Oct. 1997) 435-443.

[38] Mattax, C. C., and Dalton, R.L.: *Reservoir Simulation*, Society of Petroleum Engineers, Monograph (1977), vol. 13.

[39] Meyer, B.: *Object-Oriented Software Construction*, Prentice Hall International (1988).

[40] Mucharam, L. and Adewumi, M.A.: “A Compositional Two-Phase Flow Model for Analyzing and Designing Complex Pipeline Network Systems,” SPE paper 21562 (June 1990).

[41] Musser, D.R. and Saini, A.: *STL Tutorial and Reference Guide: C++ Programming with the Standard Template Library*, Addison-Wesley Professional Computing Series (1996).

[42] Nacul, E.C.: "Use of Domain Decomposition and Local Grid Refinement in Reservoir Simulation", Ph.D. Dissertation, Stanford University, March 1991.

[43] Nogaret, C.: "Implementation of a Network-Based Approach in an Object Oriented Reservoir Simulator", M.S. Thesis, Stanford University, May 1996.

[44] Nolan, J.S.: "Treatment of Wells in Reservoir Simulation," presented at the Third International Forum on Reservoir Simulation, Baden, Austria, July 23-27, 1990.

[45] Ouyang, L-B. and Aziz, K.: "Steady-State Gas Flow in Pipe", *Journal of Petroleum Science and Engineering* (1996), vol. 14, No. 1, 137-158.

[46] Ouyang, L-B.: "Single Phase and Multiphase Fluid Flow in Horizontal Wells", Ph.D. Dissertation, Stanford University, August 1998.

[47] Palagi, C.L.: “Generation and Application of Voronoi Grids to Model Flow in Heterogeneous Reservoirs,” Ph.D. Dissertation, Stanford University, May 1992.

[48] Parashar, M., J. Wheeler, G. Pope, K. Wang, and P. Wang. "A New Generation EOS Compositional Reservoir Simulator: Part I–Formulation and Discretization," and "Part II–Framework and Multiprocessing," presented at the Society of

Petroleum Engineers Reservoir Simulation Symposium, Dallas, Texas, June 8–11, 1997.

[49] Peaceman, D.: *Fundamentals of Numerical Reservoir Simulation*, Elsevier Scientific Publishing Company (1977).

[50] Peaceman, D.W.: “A Nonlinear Stability Analysis for Difference Equations Using Semi-Implicit Mobility,” *SPEJ* (Feb. 1977), 79-91.

[51] Peaceman, D.W.: “Discussion of an Adaptive Implicit Switching Criterion Based on Numerical Stability Analysis”, *SPER* (May 1989), 255-256.

[52] Peaceman, D.W.: “Interpretation of Well-Block Pressure in Numerical Reservoir Simulation”, *SPEJ* (June 1978), 183-194.

[53] Peaceman, D.W.: “Interpretation of Well-Block Pressure in Numerical Reservoir Simulation with Nonsquare Grid Blocks and Anisotropic Permeability,” *SPEJ* (June 1993), 531-543.

[54] Pedrosa, O. A., Jr.: "Use of Hybrid Grid in Reservoir Simulation", Ph.D. Dissertation, Stanford University (Dec. 1984).

[55] Richtmyer, R.D: *Difference Methods for Initial-Value Problems*, Interscience Publishers Inc., New York (1957), Chapter 10.

[56] Ros, N.C.J.: "An Analysis of Critical Simultaneous Gas-Liquid Flow Through a Restriction and its Application to Flow Metering," *Applied Sci. Research,* (1960), 2, pp. 374.

[57] Rumbaugh, J., Blaha, M., Premerlani, W. and Lorenson W.: *Object Oriented Modeling and Design*, Prentice-Hall, Englewood Cliffs, (1991).

[58] Russell, T.F., “Stability Analysis and Switching Criteria for Adaptive Implicit Methods Based on the CFL Condition” SPE paper 18416 (Feb. 1989), 97-106.

[59] Saad, Y.: *Iterative Methods for Sparse Linear Systems*, PWS Kent (1995).

[60] Sachdeva, R., Schmidt, Z., Brill, J. P., and Blais, R. M.: "Two-Phase Flow through Chokes," SPE paper 15667, presented at 61st SPE Fall Conf. New Orleans, LA, 1986.

[61] Schiozer, D.J.: "Simultaneous Simulation of Reservoir and Surface Facilities," Ph.D. Dissertation, Stanford University, March 1994.

[62] Shamir, U. and Howard, C. D.: "Water Distribution Systems Analysis," *Journal of the Hydraulics Division* (Jan. 1968), ASCE, vol. 94, 219-234.

[63] Shaw, R., and Byer, T.J.: Object-Oriented Design of Surface Facilities Model using Design Patterns (unpublished), 1998.

[64] Smith, B., Bjorstad, P., and Gropp, W.: *Domain Decomposition, Parallel Multilevel Methods for Elliptic Partial Differential Equations*, Cambridge University Press (1996).

[65] Startzman, G. A., Barnes, and Wong, S.A.,: "Applications of a Microcomputer Facilities Network Modeling Program," SPE paper 16506 (June 1987).

[66] Stone, H.L.: "Estimation of Three-Phase Relative Permeability and Residual Oil Data," Journal of Canadian Petroleum Technology (Oct.-Dec. 1973), 12, 53-61.

[67] Stright, D.H., Aziz, K., Settari, A., and Starratt, F.: "Carbon Dioxide Injection into Bottom-Water, Undersaturated Viscous Oil Reservoirs," *Journal of Petroleum Technology* (Oct. 1973) 29, 1248-1258.

[68] Stroustrup, B.: *The C++ Programming Language*, Addison-Wesley Publishing Company (1987).

[69] Thomas, G.W.: *Principle of Hydrocarbon Reservoir Simulation*, International Human Resources Development Corporation (1982).

[70] Thomas, G.W. and Thurnau, D.H.: "Reservoir Simulation Using an Adaptive Implicit Method," *SPEJ* (Oct. 1983) 759-68.

[71] Verma, S.: Flexible Grids for Reservoir Simulation, Ph.D. Dissertation, Stanford University, June 1996.

[72] Wallis, G.B.: *One-Dimensional Two-Phase Flow*, McGraw-Hill (1969) first edition.

[73] Wallis, J.R. and Nolen, J.S.: “Efficient Iterative Linear Solution of Locally Refined Grids Using Algebraic Multilevel Approximate Factorizations,” SPE (Feb. 1993) 59-68.

[74] Wallis, J.R., Foster, J.A., and Kendal, R.P.: “A New Parallel Iterative Linear Solution Method for Large-Scale Reservoir Simulation”, SPE paper 21209 (Feb 1991) 84-92.

[75] Wasserman, M.L.: “Local Grid Refinement for Three-Dimensional Simulators,” SPE paper 16013 (Feb. 1987).

[76] Wiener, R.S., and Pinson, L.J.: An Introduction to Object-Oriented Programming and C++, Addison-Wesley Publishing Company (1988).

[77] Wood, D.J., and Charles, C.: “Hydraulic Network Analysis Using Linear Theory Simulators,” *Jo. Hydraulics Div.*, Proc. Amer. Soc. Civil Engineers (1972), vol. 98, 1157-1170.

Appendix A

Simplified Network Solution Method

A.1 Network Solution Method

This appendix presents an analysis of the network solution method used by Schiozer (1991) and demonstrates that for problems with two-phase pipeflow, the method can result in convergence problems due to incorrect Jacobian terms. In a coupled system, additional equations are required at the pipe junctions to maintain continuity principles. To simplify the numerical implementation for the coupled system a technique that reduces the number of equations required to formulate the pipe network is employed. The reduction technique is analyzed in the context of the coupled reservoir and surface facility model shown in Fig A.1.

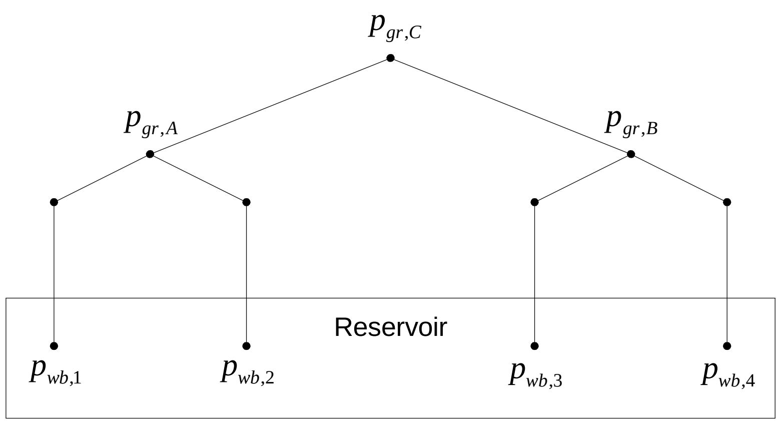


Figure A.1: Example Network

Treatment of Boundary Conditions

Only constant pressure boundaries are considered in the facility model. A residual equation is used for implementation of the network boundary condition and is expressed as

$$R_S = \left(p_{gr,sp} - p_{gr,C} \right) = 0 \tag{A.1}$$

where $p_{gr,sp}$ is the specified pressure condition at node *C* and $p_{gr,C}$ is an artificial unknown which at convergence, equals $p_{gr,sp}$.

Reduction Method

In the facility network, pipe segments are identified as pipe sections between a well and junction node or between two junction nodes. For each pipe segment the following residual equation is formulated:

$$R_i = \left(p_o - p_i + \Delta p_{io} \right) = 0 \tag{A.2}$$

where Δp_{io} is the pressure drop from node *i* to *j*, and p_i and p_o are the inlet and outlet pressures, respectively. The reduction procedure can be demonstrated by considering the flow path connecting the first well and group node *C* shown in Fig. A.1. Noting that $\Delta p_{1C} = \Delta p_{1A} + \Delta p_{AC}$, the intermediate group *A* pressure unknown can be eliminated by combining

$$R_1 = \left(p_{gr,C} - p_{gr,A} + \Delta p_{AC} \right) = 0 \tag{A.3}$$

and

$$R_2 = \left(p_{gr,A} - p_{wb,1} + \Delta p_{1A} \right) = 0 \tag{A.4}$$

to obtain

$$\tilde{R}_2 = \left(p_{gr,C} - p_{wb,1} + \Delta p_{1C} \right) = 0 \tag{A.5}$$

The new residual equation involves only pressures at the well nodes and facility boundary. A general form of the residual equation for this example is given by

$$\tilde{R}_i = \left(p_{gr,C} - p_{wb,i} + \Delta p_{iC} \right) = 0 \tag{A.6}$$

where the pressure drop Δp_{iC}, is computed using multiphase flow correlations using current iteration node pressure estimates.

In this approach only one equation per well is required and the Jacobian construction for the facility terms has been simplified. However, it is shown next that this type of reduction may introduce convergence problems in the Newton iteration.

A.2 Analysis of Jacobian Terms

The problem associated with the reduced set of residual equations is established by examining the Newton iteration (υ) for Eq. A.5 versus an equivalent form given by

$$\tilde{R}_e = \left(p_{gr_C} - p_{wb,i} + \Delta p_{1A} + \Delta p_{AC} \right) = 0 \tag{A.7}$$

The Newton Eqs. A.5 and A.7 are given by

$$\tilde{R}_2^{\upsilon+1} = \tilde{R}_2^{\upsilon} + \frac{\partial \tilde{R}_2}{\partial p_{gr,C}} \Delta p_{gr,C} + \frac{\partial \tilde{R}_2}{\partial p_{wb,1}} \Delta p_{wb,1} \tag{A.8}$$

and

$$\tilde{R}_e^{\upsilon+1} = \tilde{R}_e^{\upsilon} + \frac{\partial \tilde{R}_e}{\partial p_{gr,C}} \Delta p_{gr,C} + \frac{\partial \tilde{R}_e}{\partial p_{wb,1}} \Delta p_{wb,1} \tag{A.9}$$

Since these equations are equivalent, the derivative terms should be equivalent and therefore require the same number of Newton iterations for convergence. The coefficients in Eq. A.8 are given by

$$\begin{aligned}\frac{\partial \tilde{R}_2}{\partial p_{gr,C}} &= 1 + \frac{\partial\left(\Delta p_{1C}\right)}{\partial p_{gr,C}} \\ \frac{\partial \tilde{R}_2}{\partial p_{wb,1}} &= -1 + \frac{\partial\left(\Delta p_{1C}\right)}{\partial p_{wb,1}}\end{aligned} \tag{A.10}$$

Similarly in Eq. A.9 the coefficients are given by

$$\begin{aligned}\frac{\partial \tilde{R}_e}{\partial p_{gr,C}} &= 1 + \frac{\partial\left(\Delta p_{AC}\right)}{\partial p_{gr,C}} \\ \frac{\partial \tilde{R}_e}{\partial p_{wb,1}} &= -1 + \frac{\partial\left(\Delta p_{1A}\right)}{\partial p_{wb,1}}\end{aligned} \tag{A.11}$$

Comparing A.10 and A.11 shows that computation of the Jacobian terms based on the equation reduction method requires that

$$\begin{aligned}\frac{\partial\left(\Delta p_{1C}\right)}{\partial p_{gr,C}} &= \frac{\partial\left(\Delta p_{AC}\right)}{\partial p_{gr,C}} \\ \frac{\partial\left(\Delta p_{1C}\right)}{\partial p_{wb,1}} &= \frac{\partial\left(\Delta p_{1A}\right)}{\partial p_{wb,1}}\end{aligned} \tag{A.12}$$

Under multiphase flow conditions, application of the coefficients in Eq. A.12 represent an assumption that the derivatives can be linearly extrapolated. This can result in an excessive number of Newton iterations, which leads to convergence failure due to maximum iteration limits.

Appendix B

Auxiliary Test Model Data

B.1 Reservoir Fuild and Rock Properties

Table B.1: Fluid Properties

p psia	B_o bbl/STB	B_w bbl/STB	B_g bbl/scf	B_w bbl/STB	μ_o cp	μ_w cp	μ_g cp	R_s scf/STB
14.7	1.062	1.0410	0.166666	1.062	1.040	0.31	0.0080	1.0
264.7	1.150	1.0403	0.012093	1.150	0.975	0.31	0.0096	90.5
514.7	1.207	1.0395	0.006274	1.207	0.910	0.31	0.0112	180.0
1014.7	1.295	1.0380	0.003197	1.295	0.830	0.31	0.0140	371.0
2014.7	1.435	1.0350	0.001614	1.435	0.695	0.31	0.0189	636.0
2514.7	1.500	1.0335	0.001294	1.500	0.641	0.31	0.0208	775.0
3014.7	1.565	1.0320	0.001080	1.565	0.594	0.31	0.0228	930.0
4014.7	1.695	1.0290	0.000811	1.695	0.510	0.31	0.0268	1270.0
5014.7	1.827		0.000649	1.827	0.449	0.31	0.0309	1618.0
9014.7	2.357	1.0130	0.000386	2.357	0.203	0.31	0.0470	2984.0

Table B.2: Rock and Fluid Data Rock and Fluid Data

$\rho_g^{std}\ (lbm/ft^3)$	0.0647	$\mu_g^{std}\ (cp)$	0.008
$\rho_w^{std}\ (lbm/ft^3)$	62.238	$\mu_w^{std}\ (cp)$	0.31
$\rho_o^{std}\ (lbm/ft^3)$	42.244	$\mu_o^{std}\ (cp)$	1.04
$c_r\ (psia^{-1})$	$3(10)^{-6}$	$c_o\ (psia)^{-1}$	$1.3687(10)^{-5}$

Table B.3: Oil-Water Relative Permeability Data

S_w	k_{rw}	k_{row}
0.12000	0.00000	1.00000
0.13000	0.00000	1.00000
0.92000	1.00000	0.00000

Table B.4: Oil-Gas Relative Permeability Data

S_g	k_{rg}	k_{rog}
0.00000	0.00000	1.00000
0.00100	0.00000	1.00000
0.02000	0.00000	0.99700
0.05000	0.00500	0.98000
0.12000	0.02500	0.70000
0.20000	0.07500	0.35000
0.25000	0.12500	0.20000
0.30000	0.19000	0.09000
0.40000	0.41000	0.02100
0.45000	0.60000	0.01000
0.50000	0.72000	0.00100
0.60000	0.87000	0.00010
0.70000	0.94000	0.00000
0.85000	0.98000	0.00000
1.00000	1.00000	0.00000

Appendix C

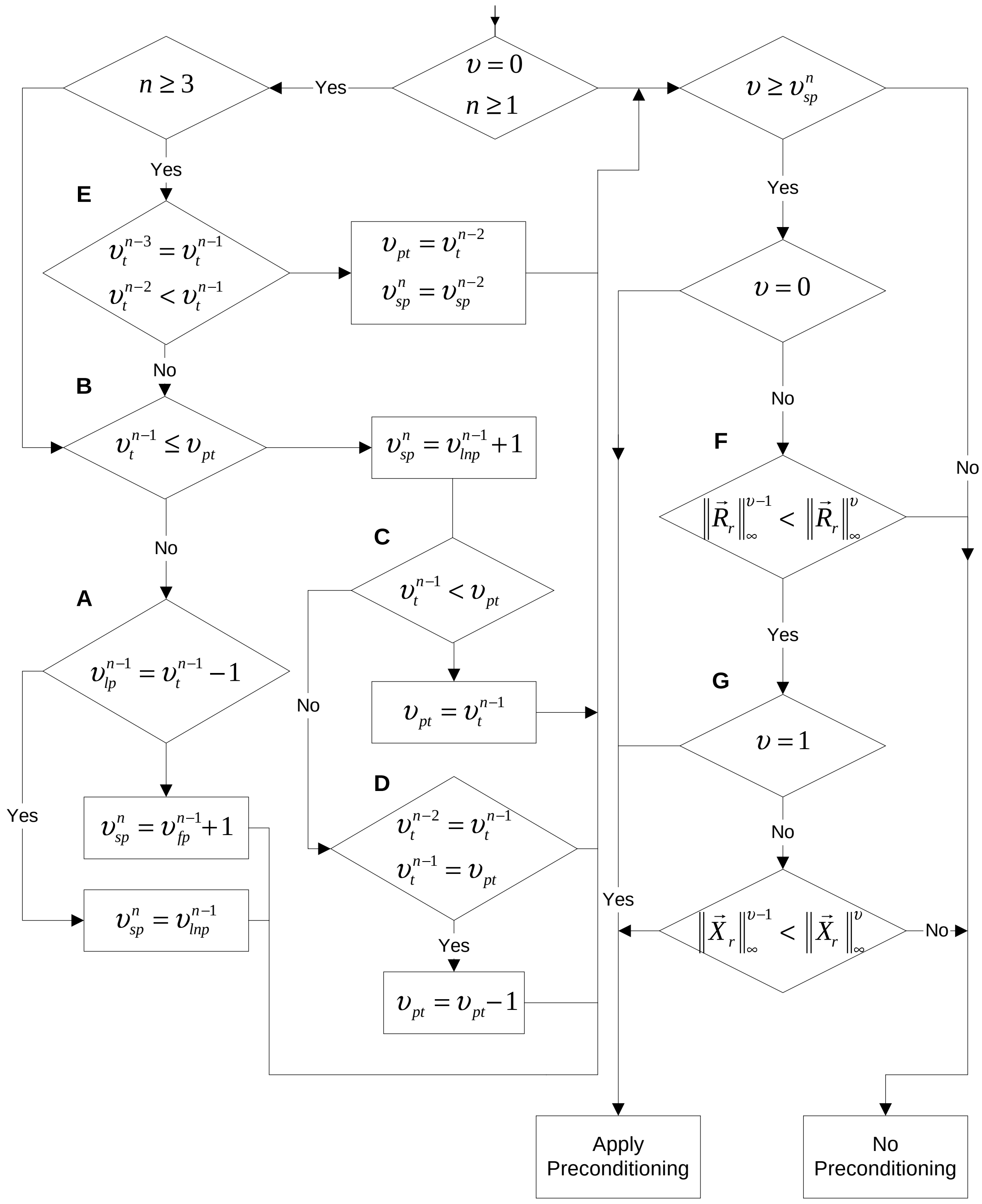
Variable Preconditioning Strategy

C.1 Description

The variable preconditioning strategy developed for this research employs multiple criteria applied over a timestep to determine whether preconditioning should be applied. The criteria are defined in terms of material balance errors, Newton iteration convergence behavior, and the performance of the previous iterations preconditioning strategy. The strategy is presented in Fig. C.1 and the flowchart variables are defined as:

υ	- current Newton iteration number of standard implicit problem
υ_t^n	- total Newton iterations for timestep *n*
υ_{fp}^n	- first preconditioned Newton iteration for timestep *n*
υ_{lnp}^n	- last nonpreconditioned Newton iteration for timestep *n*
υ_{lp}^n	- last preconditioned Newton iteration for timestep *n*
υ_{sp}^n	- first iteration at which preconditioning can be considered for timestep *n*
υ_{pt}	- preconditioning threshold iteration number

The flowchart contains several decision points that can be categorized into two general types of criteria defined by the use of either previous (left side of diagram) or current timestep information (right side of diagram). Each of these criteria are defined in



$\upsilon = 0$, $n \geq 1$
Yes
$n \geq 3$
Yes
E
$\upsilon_t^{n-3} = \upsilon_t^{n-1}$, $\upsilon_t^{n-2} < \upsilon_t^{n-1}$
$\upsilon_{pt} = \upsilon_t^{n-2}$, $\upsilon_{sp}^n = \upsilon_{sp}^{n-2}$
No
B
$\upsilon_t^{n-1} \leq \upsilon_{pt}$
$\upsilon_{sp}^n = \upsilon_{lnp}^{n-1} + 1$
No
A
$\upsilon_{lp}^{n-1} = \upsilon_t^{n-1} - 1$
Yes
$\upsilon_{sp}^n = \upsilon_{fp}^{n-1} + 1$
$\upsilon_{sp}^n = \upsilon_{lnp}^{n-1}$
C
$\upsilon_t^{n-1} < \upsilon_{pt}$
No
$\upsilon_{pt} = \upsilon_t^{n-1}$
D
$\upsilon_t^{n-2} = \upsilon_t^{n-1}$, $\upsilon_t^{n-1} = \upsilon_{pt}$
Yes
$\upsilon_{pt} = \upsilon_{pt} - 1$
$\upsilon \geq \upsilon_{sp}^n$
Yes
$\upsilon = 0$
No
F
$\left\|\vec{R}_r\right\|_\infty^{\upsilon-1} < \left\|\vec{R}_r\right\|_\infty^{\upsilon}$
Yes
G
$\upsilon = 1$
No
$\left\|\vec{X}_r\right\|_\infty^{\upsilon-1} < \left\|\vec{X}_r\right\|_\infty^{\upsilon}$
Yes
No
No
Apply Preconditioning
No Preconditioning

Figure C.1: Preconditioning Frequency Strategy

detail, beginning with the core decision logic. Note that in Fig. C.1 timestep and Newton iteration indexing is relative to 0, following the C/C++ coding conventions. However, the number of iterations required for a timestep $\left(\upsilon_t^n\right)$ and threshold value $\left(\upsilon_{pt}\right)$ are absolute totals.

The criteria based on current timestep information are used to control the application of the preconditioner within the current timestep. The criteria are based on the premise that application of the preconditioner should not increase material balance errors or delay Newton convergence rate, thus preconditioning will only occur if the material balance error and/or Newton solution vector are in a descent direction.

The selection of the first iteration at which preconditioning is considered for the current timestep is based on performance data of the preconditioning strategy at earlier timesteps. Application of the criteria described in Fig. C.1 yields a sequence of Newton iterations that can be classified as either preconditioned or nonpreconditioned. The criteria developed using this data is explained by examining possible preconditioning sequences defined by $\left\{pc^{\upsilon},\ \upsilon = 0,1,2,...\right\}$, where $pc^{\upsilon} = 1$ if iteration υ was preconditioned, otherwise $pc^{\upsilon} = 0$. With respect to the decision logic shown in Fig. C.1, a succession of criteria is defined as follows:

1. This criterion is designed to delay preconditioning. The condition $\upsilon_{lp}^{n-1} \neq \upsilon_t^{n-1} - 1$ (decision point A) is for sequences such as $\left\{0,0,...,0\right\}$ or $\left\{1,0,1,0,..,0\right\}$ where preconditioning was either not started, or was started, disabled, and never resumed. This implies that application of the preconditioner may have been of limited benefit since the reservoir conditions were changing later in the iterations. Therefore on the current iteration this criterion will delay the possibility of preconditioning by one iteration with respect to the first preconditioned iteration of the previous timestep, υ_{fp}^{n-1}.
2. This criterion seeks to find the earliest iteration that preconditioning can begin and continue uninterrupted. The condition corresponding to

$\upsilon_{lp}^{n-1} = \upsilon_{t}^{n-1} - 1$ (decision point A) is designed for sequences of the form $\{0,..,0,1,1\}$ or $\{1,1,0,..,0,1,1\}$, where early in the iterations preconditioning was either disabled or delayed and continued uninterrupted only towards the end of the iterations. In this case the current iteration cannot begin preconditioning until the iteration number corresponding to the last nonpreconditioned iteration of the previous timestep υ_{lnp}^{n-1}.

Criteria selection is determined via the preconditioning threshold parameter, υ_{pt}, which defines an acceptable number of Newton iterations per timestep. If this parameter is exceeded, $\upsilon_{t}^{n-1} > \upsilon_{pt}$, and then modifications to the strategy are employed that are based on the first two criteria. Otherwise, the current strategy is considered successful and steps are taken to maintain or improve the convergence rate. The threshold parameter initially assumes a specified value and then under various conditions, can change with convergence behavior. These conditions are discussed with respect to Fig. C.1 and help further define υ_{pt}.

1. The condition $\upsilon_{t}^{n-1} \leq \upsilon_{pt}$ (decision point B) indicates the previous number of Newton iterations required for convergence is less than or equal to the threshold value. The previous timestep's preconditioning strategy coupled with reservoir behavior resulted in an acceptable convergence rate as defined by the threshold value. Therefore the first iteration at which preconditioning will be considered is set to one iteration beyond the last nonpreconditioned iteration of the previous timestep, $\upsilon_{sp} = \upsilon_{lnp}^{n-1} + 1$.
2. The more restrictive condition, $\upsilon_{t}^{n-1} < \upsilon_{pt}$ (decision point C) indicates that the previous number of Newton iterations required for convergence is less than the threshold value. The previous timesteps preconditioning strategy coupled with reservoir behavior resulted in accelerated convergence rate and defines a new goal for the preconditioner. Therefore the threshold value is

set equal to the number of iterations required for the previous timestep, $\upsilon_{pt} = \upsilon_t^{n-1}$.

3. The condition $\upsilon_t^{n-2} = \upsilon_t^{n-1}$ (decision point D) is designed to prevent the strategy from operating at a threshold value for more than two iterations. In this case the threshold value is decreased $\upsilon_{pt} = \upsilon_{pt} - 1$. This criterion helps the strategy continually test starting points for preconditioning.

4. The conditions $\upsilon_t^{n-3} = \upsilon_t^{n-1}$ and $\upsilon_t^{n-2} < \upsilon_t^{n-1}$ (decision point E) are designed to detect oscillating behavior in the Newton convergence rate over the last three timesteps. Note that timestep *n-2* required the least iterations. If this condition is satisfied, in an attempt to regain the better convergence rate, the first iteration in which preconditioning can be considered, υ_{sp}^n, is set equal to the value used at timestep *n-2*, υ_{sp}^{n-2}. Also the preconditioning threshold is reset to the number of iterations required for timestep *n-2*. For low threshold values this has the effect of delayed preconditioning and for high threshold values this has the effect of considering preconditioning earlier than before.

C.2 Sample Data

The variable preconditioning strategy is demonstrated with four sets of preconditioning sequences that correspond to 4 timesteps. The data are not from an actual run, however are representative of observed behavior. The sequences are presented in the following Table C.1. Recall that $pv^{\upsilon} = 1$ indicates that the iteration is preconditioned, otherwise $pv^{\upsilon} = 0$ and a standard implicit iteration is performed. The initial preconditioning threshold is initially set to four $\left(\upsilon_{pt} = 4\right)$ and preconditioning is automatically applied on the first iteration of the simulation run $\left(\upsilon_{sp}^0 = 0\right)$. Note that since $\upsilon_{sp}^0 = 0$, preconditioning will be considered at all iterations of the first timestep.

Table C.1: Sample Preconditioning Performance Data

	pc^{υ}					
Newton Iteration, υ	0	1	2	3	4	5
Timestep Number, n						
0	1	0	0	1	1	1
1	0	0	1	1	0	-
2	0	0	0	1	-	-
3	0	0	0	1	-	-

Timestep Number, n	υ_{sp}^{n}	υ_{pt}^{n}	υ_{t}^{n}	υ_{fp}^{n}	υ_{lp}^{n}	υ_{lnp}^{n}
0	0	4	6	0	5	2
1	2	4	5	2	3	4
2	3	4	4	3	3	2
3	3	4	4	2	2	1

Timestep *n=0*

The first timestep required 6 Newton iterations for convergence. Preconditioning is applied the on first iteration and subsequently disabled until $\upsilon = 3$, due to a non-decent direction in the material balance error and/or linear solution vector (decision points F, G). At $\upsilon = 3$ preconditioning was activated and applied every iteration for the remainder of the timestep.

Timestep *n=1*

The total number of iterations required for the previous iteration $\left(\upsilon_t^0 = 6\right)$ exceeded the threshold value $\left(\upsilon_{pt} = 4,\ \text{decision point B}\right)$ and therefore logic beginning at decision point A is used to establish when preconditioning can be considered. Since the last iteration of the previous timestep was preconditioned, the first iteration at which preconditioning can be considered is equal to the last nonpreconditioned iteration of the previous timestep, $\left(\upsilon_{sp}^{1}=\upsilon_{lnp}^{0}\right)$. In this case the strategy is exploring how early preconditioning can be started. Preconditioning is applied for iterations 2 and 3, and then disabled due to a non-decent direction in the material balance error and/or linear solution vector (decision points F, G). A total of 5 iterations are required for the timestep.

Timestep *n=2*

The total number of iterations required for the previous iteration $\left(\upsilon_{t}^{1}=5\right)$ still exceeds the threshold value $\left(\upsilon_{pt}=4, \text{ decision point B}\right)$ and therefore logic beginning at decision point A is used to establish when preconditioning can be considered. Since the last iteration of the previous timestep was not preconditioned, the first iteration that preconditioning can be considered is equal to one iteration beyond the first preconditioned iteration of the previous timestep, $\left(\upsilon_{sp}^{2}=\upsilon_{fp}^{1}+1\right)$. In this case, preconditioning was applied too early in the previous timestep and therefore the strategy will delay preconditioning for the current timestep. Preconditioning is applied only for iteration 3. A total of 4 iterations are required for the timestep.

Timestep *n=3*

The total number of iterations required for the previous iteration $\left(\upsilon_{t}^{2}=4\right)$ equals the threshold value $\left(\upsilon_{pt}=4, \text{ decision point B}\right)$, therefore preconditioning is considered at the one iteration beyond the last nonpreconditioned iteration of the previous timestep, $\upsilon_{sp}^{3}=\upsilon_{lnp}^{2}+1$. This corresponds to iteration 3 and reflects a strategy of maintaining the preconditioning strategy used for the previous timestep. Since the condition $\upsilon_{t}^{2}<\upsilon_{pt}$ (decision point C) is false the strategy will examine the number of iterations required by the previous two timesteps to determine whether the strategy has been operating at the threshold value. If so, then the strategy will seek to improve the convergence rate by reducing the threshold value by one iteration. This has the affect of employing the logic beginning at decision point A if the number of iterations required for the current iteration exceeds the threshold value. In this example, the threshold value is not reset since $\upsilon_{t}^{2}\neq\upsilon_{t}^{1}$.

C.3 Concluding Remarks

These criteria are designed to improve the quality of preconditioning by responding to preconditioner performance based on specified well subdomain size. The variable preconditioning criteria are independent of subdomain size and therefore this strategy is a critical component in the overall preconditioning method since determination of the optimal subdomain size requires numerous trial runs and is problem dependant.

Appendix D

Object-Oriented Concepts

D.1 Object-Oriented Terminology and Concepts

The purpose of this section is to present the basic entities in object-oriented modeling. A detailed discussion of these concepts can be found in Rumbaugh (1991) or Meyer (1992).

D.1.1 Class

A class is a description of a group of objects with similar attributes, operations, and relationships. An attribute is a characteristic or property of an object. An operation is function or set of actions. Two important default functions are the constructor and destructor. The constructor is called whenever a class is instantiated and serves to initialize or define attributes and establish relationships. The destructor is called whenever a class is deleted and serves to deallocate any resources assigned to the class. A relationship is a connection or link between classes or objects. The primary relationships are association (“has a”), aggregation (“part of”), and generalization specialization (“is a”).

D.2.1 Object

An object is an instance of a class and responds to messages from other objects. Objects are complete entities, they encapsulate the required data and concepts to define a real world object. There are three major forms of objects: atomic, structured and collection. An atomic object is an object of a primitive class or data type, e.g. integer, float, or point.

A structured object is an object of a class with attributes, operations, and relationships. A collection object is an object containing an array, list, or set.

D.3.1 Relationship

A relationship is a link or connection between two classes or between two objects. The three major types of relationships are described.

Association

An association relationship may be described as “has a”, “associated with”, or “knows about”. An association can be one to one, or one to many. An association is implemented as through dynamic membership in a class, thus reflecting a weak cohesion between classes.

Aggregation

Aggregation is a link between classes that represents a “part of “ relationship. An aggregation represents a stronger cohesion between objects than an association, and indicates an integral part of the object. An association can also have a one to one, or one to many relationship. An aggregation is implemented as a static definition in a class.

Generalization Specialization

A generalization specialization relationship indicates a commonality between a superclass and subclass, which can be expressed via common attributes, operations or relationships. This relationship is commonly identified via the “is a” type connotation. The programming implementation employs inheritance to achieve the common elements.

D.4.1 Diagram Notation

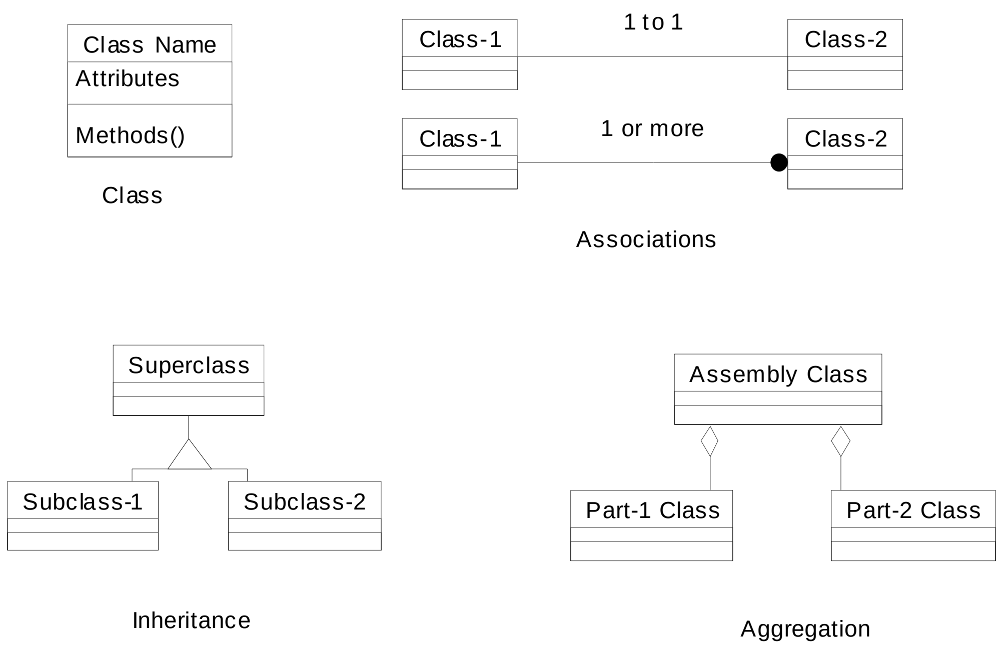


Figure D.1: Class Diagram Notations

D.2 Standard Template Library

Several class libraries are commercially available, which provide functionality such as string manipulation, matrix manipulation, data persistence, or thread management. The most significant library addition to the C++ standard is the Standard Template Library (STL). The STL is a library of container classes, algorithms, and iterators. The container classes include vectors, lists, and sets. The algorithms include sort, reverse or splice operations on the container classes. The iterators provide a standard method for accessing data in container classes. Some of these components are discussed further here. For a detailed description of the STL see Musser and Saini (1996).

Vector Container

The vector container is similar to a contiguous C array and provides array-like random access. Several support operations are available for insertion and deletion at any point in the array. The *vector* automatically resizes, though functions are provided to reserve a minimum amount of storage. The following example demonstrates the use of a vector container for integers.

```
vector<int> v(3);                // declare a vector of 3 integer elements
vector<char> s(3);               // declare a vector of 3 character elements.
v[0] = 6;                        // vector assignments
v[1] = 1;
v[2] = v[0] + v[1];              // v[0] = 6, v[1] = 1  => v[2] = 7
v.insert(v.begin()+2,3);         // insert 3 at index location  2
reverse(v.begin(), v.end());     // v[0] = 7, v[1] = 3, v[2] = 1, v[3] = 6
```

The flexibility of the vector container can be summarized with the following observations:

- the type of vector is arbitrary
- the vector can be resized dynamically
- no additional code is required in order to insert new data or perform elementary operations on the data
- the reverse function is independent of data type

Additionally the vector container is internally designed to be almost as efficient as standard data arrays with respect to operations available to standard data arrays. But inserting into the middle of a vector is expensive, because elements must be moved down, and growing the vector is costly because it must be copied to another vector internally.

List Container and Iterators

The list container is similar to a doubly linked list. The list differs from the vector class in two main areas. The list does not provide random access to the data elements, however it is more efficient when adding or deleting an item in the middle of the list. The following example demonstrates the use of the list container and iterators.

```
double array1 [] = { 9.6, 1.6, 3.6 };          // declare array of 3 double elements
double array2 [] = { .1, 4.5 };                // declare array of 3 double elements
list< double > D1 (array1, array1 + 3);        // create list D1 of type double and initialize
                                                  with array1 data
list< double > D2 (array2, array2 + 2);        // create list D2 of type double and initialize
                                                  with array2 data
list< double >::iterator iter = D1.begin ();   // get pointer to beginning of D1 list
iter.splice (D1, D2);                          //  add data from D2 list to beginning of  D1 list
list< double >::iterator iter = D1.begin ();   //  get pointer (iter) to beginning of D1 list
while (iter != D1.end ())                      //  while iter does not point to end of list
   cout << * iter ++ << endl;                      print list D1 contents to standard output
```

Note the iterator type specification of *D1* and *D2*. The iterator points to objects in the list container and also contains several member functions to index the data. In this example the *begin()* and *end*() member functions return pointers to the beginning and end of the *D1* list. Also an *iterator* type is used to define the input for many of the STL algorithms, for example see the *splice()* function used above. Similar to the vector container, the flexibility of the list container is demonstrated in the following areas:

- the type of list is arbitrary
- the size of the list can be dynamically resized
- no additional code is required to insert new data or perform elementary operations on the data
- the splice function is independent of data type
- the list can return an object of type iterator for use in data operations provided by the iterator and algorithm

The simple examples presented above demonstrate the generic programming capability provided by the STL. The container classes provide a strong basis for development of specialized data storage features required by simulation applications. The following example demonstrates how the list container is employed to provide access to an array of cell pressures via the request

double *opres = getDataVec("OPRES");.

```
DataVector * CartesianGrid::getData(char *aname) {     // interface to getData operation
list<DataVector*>::iterator iter = listofData.begin();  // set pointer to beginning of list
int found=1;
while (iter != listofData.end() && found != 0) {        // while the pointer is not at end of list and
                                                           data has not been found
   found = strcmp(aname,(*iter)->getname());            //  access DataVector name and compare
   ++iter;                                              //  increment pointer
        }
if ( found != 0 ) return NULL;                          //  data not found return NULL
iter--;                                                 //  backup pointer
return(*iter)->Vec();                                   //  return vector stored inside DataVector
}
```

D.3 Concluding Remarks

A brief discussion has been presented on basic object-oriented concepts. Other references that will help the reader better understand object-oriented programming using the C++ language include Lippman (1989), Stroustrup (1987), and Wiener and Pinson (1988). Also the PETSc home page (Baley *et al.* 1999) provides links to excellent discussions related to the object-oriented design of numerical toolkits.