NONLINEAR OPTIMIZATION OF WELL PRODUCTION CONSIDERING GAS LIFT AND PHASE BEHAVIOR

A REPORT

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

MASTER OF SCIENCE

By

Miles Robert Palke

September 1996

Approved for the Department:

_____

Roland N. Horne

Abstract

This report describes numerical techniques developed to determine the gas-lift configuration that will optimize the lifetime value of a property. The production history of a reservoir can be predicted by integrating a reservoir model, a wellbore flow model, a choke model, and a separator model. Changes in any production parameter, including the gas-lift configuration, will cause changes in the predicted production history. The numerical methods can find the combination of production parameters that maximizes the net present value of the flowstream. The control parameters explored include tubing diameter, separator pressures, depth of gas injection, and volume of gas injected. Each of these parameters may vary with time. The influence these parameters have upon the net present value is complicated by the feedback nature of the gas-injection loop. This nonlinearity requires robust, efficient routines for optimization. The optimizing model is used to demonstrate that both objective function (net present value), and optimal production parameter values can be very sensitive to uncertainty in the known reservoir data. This sensitivity study also illustrates a method of determining the value of additional reservoir data.

This report describes how to fashion an integrated wellbore and reservoir model, including the difficult feedback aspects of gas-lift. Additionally, insight is granted into the sensitivity of optimized parameters to uncertainty in reservoir data. Polytope and genetic algorithm optimization techniques are shown to be both stable and efficient when used to optimize these sorts of nonlinear optimization problems.

Acknowledgments

During my stay at Stanford, I have had the benefit of fine instruction and advice. Most notably, I would like to thank Dr. Roland Horne for his direction and guidance. Additionally, I would like to thank Professors Aziz and Hewett for their inspiring dedication to the advancement of the science of petroleum engineering. I would also like to thank the faculty in its entirety. Of course, it would be wrong to ignore the assistance of the staff, including, but not limited to, Sandy Costa, Stephanie, Jeanne, and Yolanda. Special, special, special thanks are due to Christina Del Villar, who was of unlimited help.

I have made many friends during my stay at Stanford, and these friends have contributed a great deal to my success. Thanks to Rosalind Archer, Willis Ambusso, Richard Williams, Pierre Maizeret, Tony Lolomari, Roman Santa Maria, Daryl Fenwick, Rob Batycky, Richard Hughes, Genevieve Conaty, and Noel Urmeneta. Of course I've omitted some, but not forgotten them. Oscar Wilde said, "It is absurd to divide people into good and bad. People are either charming or tedious." Good or bad, none of these people are tedious.

I save special thanks for other members of SUPRI-D. Jorge, Suwat, Yan, Bit, and others have provided a great deal of council, explanation, and advice. The quality and kind of work performed in SUPRI-D is truly inspirational.

No one has contributed more to my success and happiness than my family. Thank you mother, father, Neal, Dale, and Susan.

Contents

List of Tables

List of Figures

1. Introduction

Petroleum engineers face a wide variety of parameter estimation and optimization problems. Every time a transient test is analyzed or a location for a well is chosen, an optimization or parameter estimation problem has been solved. Yet it is surprising how rarely a formal optimization technique is used to solve these problems. Mathematical and heuristic methods are available which can find optimal solutions (or correct solutions for parameter estimation problems) for a given model. This report describes the application of formal optimization techniques to a realistic petroleum problem.

Many engineering solutions involve the use of models. The model may be as simple as a linear equation, or as sophisticated as a million element simulation. Regardless, all models include engineering parameters, constants, and an objective function. The engineering parameters, or decision variables, are those values that the engineer can control or modify. The constants are physical properties over which the engineer has no control, but that influence the output of the model. The objective function is the output of the model, be it profit, torque, or volume produced. Optimization is a process of selecting the engineering parameters such that the objective function is either minimized or maximized, subject to some family of constraints.

Many optimization techniques are available to solve a wide variety of problems. However, many times petroleum engineering decisions are made without the benefit of formal optimization. With the improvement of computer modeling over time, it is now feasible to apply optimization techniques to petroleum engineering models. Part of the focus of the SUPRI-D research group has been the development of optimization and parameter estimation techniques. This project represents the most recent in a series of field optimization projects.

In 1990, Carroll applied multivariate optimization techniques to a field produced by a single well. His model included a single well oil field. However, only the separator model was compositional, and no engineering parameters were allowed to vary with time. Carroll used two types of optimization routines, gradient methods and polytope methods. In 1992, Ravindran followed suit. Ravindran also allowed for gas-lift, and

allowed engineering parameters to vary with time. Fu jii followed in 1993, by allowing a network of wells connected at the surface. Fujii also studied the utility of genetic algorithms for petroleum engineering optimization.

This project expanded on Ravindran's work by replacing all black-oil components with fully compositional models. In addition, all three major types of optimization techniques were considered. Furthermore, the project studied the sensitivity of optimized parameters to uncertainty in reservoir parameters. This aspect of optimization techniques should provide insight into the value of reservoir data.

2. Model Construction

2.1 Model Description

The model developed for this project was designed as a test for the use of optimization algorithms in petroleum engineering problems. There are several criteria that prescribe a good test. In addition to being robust, the model is a simplification of a real petroleum engineering problem. If the tested optimization algorithms are to be useful for real applications, the test problem should include challenging nonlinear relationships. The field model constructed for this work meets these requirements.

The field model constructed is an integration of smaller components. The complete model represents an oil reservoir with a single gas-lifted well. The smaller model components include:

- Reservoir model;
- Well model with gas lift;
- Choke model;
- Separator Model.

Figure 2.1 illustrates the fashion in which these components are linked. All of these component models are described in more detail later in this section.

In the integrated model, each component influences the other components. Produced fluid flows into the tubing. At the point where the lifting gas enters the tubing, the two streams commingle, and the flow continues up the wellbore. At the surface, the commingled fluid passes through the choke into the first separator. Gas from the first separator goes into the gas line and the liquid phase moves into the second separator. Gas from the second separator passes into the gas line, and the liquid goes into the third separator. The gas in the third separator goes into the gas line, and the liquid goes into stock tanks to be sold. Some of the separated gas is compressed, and injected into the tubing-casing annulus for gas lift. The remainder of the gas is sold.

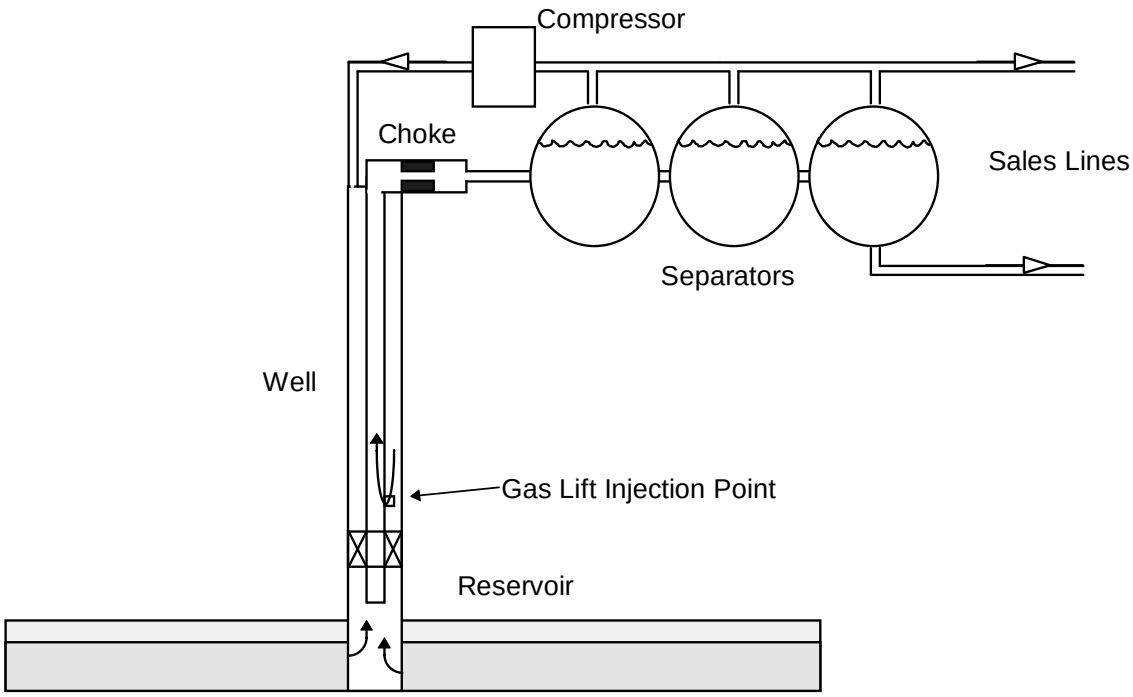

Figure 2.1: Diagram of Complete Field Model

An engineer given charge of a single gas-lift well would be expected to optimize the value or worth of that well by adjusting a variety of parameters. In this report, these parameters are referred to as decision variables, and the net present value of the well is referred to as the objective function. The decision variables in the problem include:

- Tubing diameter;
- Surface choke diameter;
- Lift gas injection rate;
- Lift gas injection depth;
- Choke diameter;
- Separator pressures.

As reservoir conditions change, the engineer would be expected to change the value of these parameters, so the model allows the decision variables to change over time. The objective function is not only dependent upon the decision variables. There are also reservoir parameters and economic factors which influence the objective function. These factors include:

1) Reservoir parameters;

- Permeability;
- Fluid composition;
- Porosity;
- Thickness;
- Areal extent of reservoir;

2) Economic factors;

- Price escalation;
- Cost escalation;
- Discount factors.

The net present value depends upon the production schedule of the well, so the model determines the twenty year production stream based on a set of decision variables and the given reservoir and economic factors. As conditions change, the production stream also changes. Notice the nonlinear relationships between the various components. The lift gas injection cycle creates a feedback loop in which the first separator pressure influences the composition and rate of inflow in various ways. Many of the subtleties of how these components interact depend upon the composition of the fluids involved. For this reason, all of the component models are based on the compositional behavior of the fluids.

2.2 Compositional Modeling

Prior optimization studies by Carroll (1990), Ravindran (1992), and Fujii (1993) used equation of state type flash calculations to model the separators. Other component models, such as the reservoir and wellbore components used a black-oil framework. This work expands the use of compositional modeling into all the component models. Compositional modeling is included to identify the sensitivity of optimization to reservoir fluid composition.

An equation of state is a model which describes the mechanical state of a substance (Walas, 1985). An equation of state, or EOS, predicts the value of dependent physical properties given the values of other independent physical properties. For this project, EOS solutions provided densities and phase compositions at given temperatures,

pressures, and mixture compositions. Using an EOS to model phase behavior eliminated the need for arcane, empirical black-oil phase behavior correlations.

For compositional modeling, the quantity and composition of each phase when a mixture of known composition is flashed at a given temperature and pressure is often needed. Also required is the density of each of the resultant phases. An EOS can be used to determine these values. The procedure to do this follows. The individual steps are described in further detail later in this section.

1) Beginning with mixture Z of n molecular components at pressure p and temperature T, assume that the gas phase composition, Y, and liquid phase composition, X, are the same as Z.
2) Use the ratio of vapor mole fraction to liquid mole fraction, or K-value for each composition to perform flash calculations. For the first step, estimate the initial K_i values with the Wilson equation (McCain, 1990). Find the liquid fractional molar volume, L, and phase compositions X and Y, for this mixture at these K_i values.
3) Use the EOS to determine the partial fugacities of each component for each phase, $\hat{f}^l$ and $\hat{f}^v$. If $\left|\frac{\hat{f}^l}{\hat{f}^v} - 1\right| \leq \varepsilon$, the EOS has converged.
4) If the process has not converged, each ratio K_i must be updated with the partial fugacities. Return to the second step.

When this method has converged, the EOS has predicted all the physical properties for the mixture at the given pressure and temperature.

2.2.1 Flash Calculations

Flash calculations provide a way of determining the liquid mole fraction, L, and phase compositions, X and Y, given a set of vapor mole fraction to liquid mole fraction ratios. These ratios are also known as K-values. The calculations are based on the following relationships:

$$K_i = \frac{Y_i}{X_i} \tag{2.1}$$

$$Z_i = LX_i + VY_i \tag{2.2}$$

$$L + V = 1 \Leftrightarrow V = 1 - L \tag{2.3}$$

So for each component of the mixture, the following equations must hold:

$$X_i = \frac{Z_i}{L + (1 - L)K_i} \tag{2.4}$$

$$Y_i = \frac{K_i Z_i}{L + (1 - L)K_i} \tag{2.5}$$

Additionally, there are the constraints:

$$\sum_{i=1}^{n} Z_i = \sum_{i=1}^{n} X_i = \sum_{i=1}^{n} Y_i = 1 \tag{2.6}$$

Two of these constraints can be combined to yield:

$$\sum_{i=1}^{n} X_i - \sum_{i=1}^{n} Y_i = 0 \tag{2.7}$$

Hence the correct solution is the root of the following function:

$$F(L) = \sum_{i=1}^{n} \frac{Z_i\left(1 - K_i\right)}{K_i + \left(1 - K_i\right)L} \tag{2.8}$$

In practice, the root, $F(L) = 0$, is found with the Newton-Raphson technique where

$$L_{k+1} = L_k - \frac{F\left(L_k\right)}{\left.\frac{\partial F}{\partial L}\right|_{L_k}} \tag{2.9}$$

and

$$\frac{\partial F}{\partial L} = -\sum_{i=1}^{n} \frac{Z_i\left(1 - K_i\right)\left(1 - K_i\right)}{\left(K_i + \left(1 - K_i\right)L\right)^2} \tag{2.10}$$

Using these equations, an iterative search locates the appropriate L, X_i, and Y_i values.

2.2.2 EOS Partial Fugacities and K-Value Factors

After iterating in the flash calculations, the values of L, X_i, and Y_i are only correct for the assumed values of K_i. However, K_i values also depend on the compositions of the phases. Thus, another iterative process is necessary.

Before the first flash calculation process, an initial estimate of each K_i value is required. This estimate is provided by the Wilson equation.

$$K_i = \frac{e^{\left[5.37(1+\omega_i)\left(1-\frac{1}{T_{ri}}\right)\right]}}{P_{ri}} \tag{2.11}$$

where

$$T_{ri} = \frac{T}{T_{ci}} \tag{2.12}$$

and

$$P_{ri} = \frac{P}{P_{ci}} \tag{2.13}$$

However, this is only valid for the first set of flash calculations. After the flash calculation convergence has been reached, the K_i values must be updated with the EOS. The EOS will also provide the densities of the phases.

This EOS used in this project was proposed by Redlich and Kwong (1949). This is a cubic EOS with the standard form shown below.

$$P = \frac{RT}{V - b_m} - \frac{a_m}{\sqrt{T}V\left(V + b_m\right)} \tag{2.14}$$

Where:

$$a_i = 0.42748 \frac{R^2 T_{ci}^{2.5}}{P_{ci}}; \quad (2.15)$$

$$b_i = 0.08664 \frac{RT_{ci}}{P_{ci}} \; ; \quad (2.16)$$

$$a_{ij} = \sqrt{a_i a_j} \; ; \quad (2.17)$$

$$a_m = \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j a_{ij} \; ; \quad (2.18)$$

$$b_m = \sum_{i=1}^{n} y_i b_i \, . \quad (2.19)$$

It is convenient to rewrite the standard form to solve for volume, given pressure and temperature. This leads to the following cubic form:

$$V^3 - \frac{RT}{P} V^2 + \frac{1}{P}\left(\frac{a_m}{\sqrt{T}} - b_m RT - P b_m^{\,2} \right) V - \frac{a_m b_m}{P\sqrt{T}} = 0 \quad (2.20)$$

Equation 2.20 can be solved once a_m and b_m have been calculated. This will yield the phase specific volumes. An analytic cubic solution is used to find the roots.

$$x^3 + px^2 + qx + r = 0 \quad (2.21)$$

To solve this, determine α and β.

$$\alpha = \frac{3q - p^2}{3} \quad (2.22)$$

$$\beta = \frac{2p^3 - 9pq + 27r}{27} \quad (2.23)$$

The number of roots can be determined as follows:

$\frac{\beta^2}{4}+\frac{\alpha^3}{27}>0$ *1 real, 2 imaginary roots;*

$\frac{\beta^2}{4}+\frac{\alpha^3}{27}=0$ *3 real, 2 equal roots;*

$\frac{\beta^2}{4}+\frac{\alpha^3}{27}<0$ *3 real roots.*

The roots (x_1, x_2, x_3) are then:

$$x_1 = A + B - \left(\frac{p}{3}\right); \tag{2.24}$$

$$x_2 = -\frac{A+B}{2} + \frac{A-B}{2}\sqrt{-3} - \left(\frac{p}{3}\right); \tag{2.25}$$

$$x_3 = -\frac{A+B}{2} - \frac{A-B}{2}\sqrt{-3} - \left(\frac{p}{3}\right). \tag{2.26}$$

where

$$A = \left[-\frac{\beta}{2} + \sqrt{\frac{\beta^2}{4}+\frac{\alpha^3}{27}}\right]^{\frac{1}{3}} \tag{2.27}$$

$$B = \left[-\frac{\beta}{2} - \sqrt{\frac{\beta^2}{4}+\frac{\alpha^3}{27}}\right]^{\frac{1}{3}} \tag{2.28}$$

It should be noted that the specific volume and density predicted by this EOS for the liquid phase will not always be appropriate. For this project, liquid phase densities were generated with an empirical method from McCain (1990).

The EOS also predicts the partial fugacity of each phase. The partial fugacity, $\hat{f}_i^{\,p}$ represents the chemical potential of component i in phase p of a mixture at a given thermodynamic state (Thiele, 1995). When the partial fugacity is equal in each phase,

for each component i, thermodynamic equilibrium has been reached. Partial fugacity has the units of pressure, so the dimensionless partial fugacity coefficient, $\hat{\phi}_i^p$ is defined for each phase as follows:

$$\hat{\phi}_i^v = \frac{\hat{f}_i^v}{Y_i p} \tag{2.29}$$

$$\hat{\phi}_i^l = \frac{\hat{f}_i^l}{X_i p} \tag{2.30}$$

The Redlich-Kwong EOS provides all partial fugacity coefficients;

$$\ln\hat{\phi}_i = \frac{b_i}{b_m}(z-1) - \ln\left[z\left(1-\frac{b_m}{V}\right)\right] + \frac{1}{b_m RT^{1.5}}\left[\frac{a_m b_i}{b_m} - 2\sqrt{a_m a_i}\right]\ln\left(1+\frac{b_m}{V}\right) \tag{2.31}$$

Bearing in mind that the pressure is the same for each phase, the following equality holds:

$$\frac{\hat{\phi}_i^l}{\hat{\phi}_i^v} = \frac{\hat{f}_i^l}{\hat{f}_i^v}\frac{Y_i}{X_i} \tag{2.32}$$

At equilibrium, $\frac{\hat{f}_i^l}{\hat{f}_i^v}$ goes to unity, and $\frac{Y_i}{X_i}$ is the same as the K_i value. If $\frac{\hat{f}_i^l}{\hat{f}_i^v}$ is near unity for each component i, then the entire process is considered to have converged. If the process has not converged, the K_i values are updated with the following relationship:

$$K_i^{k+1} = \frac{\hat{f}_i^L}{\hat{f}_i^v} K_i^k \tag{2.33}$$

At this point, the flash correlations are repeated using the updated K_i values.

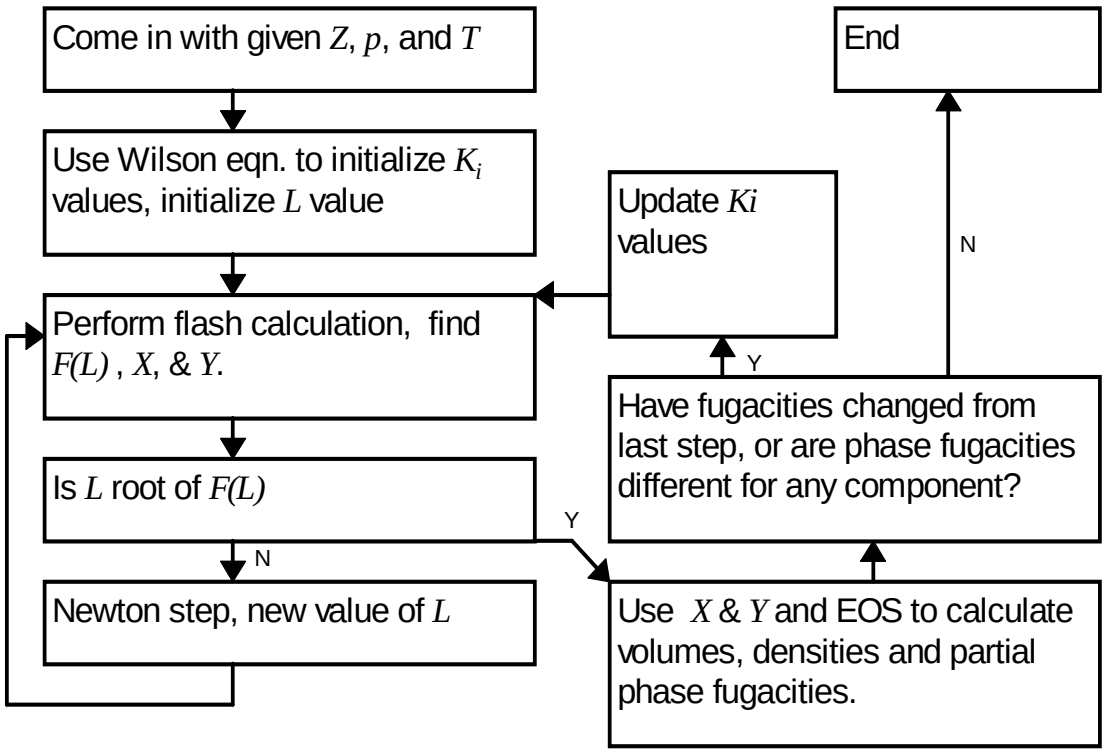

Figure 2.2: Diagram of the EOS Compositional Modeling Process

2.3 Reservoir Component

The reservoir component model developed for this project is a simple, compositional, "tank" model. The model is limited by the following assumptions:

- The reservoir is homogeneous, isotropic, cylindrical;
- The reservoir is horizontal and of constant thickness;
- The boundaries of the reservoir are no-flow;
- There is no aqueous phase, and the rock phases is incompressible;
- Both hydrocarbon phases are produced as if in pseudosteady state;
- The reservoir is produced from a single well only;
- The effects of gravity and capillary pressure are neglected.

2.3.1 Flow Rates

At any time, production of fluid is governed by the pseudosteady-state flow equation,

$$q_p = \frac{0.00708 k k_{rp} h}{\mu_p}\left[\frac{\bar{p} - p_{wf}}{\ln(r_e/r_w) - 0.75}\right] \tag{2.34}$$

For each phase, flow from the reservoir is governed by Equation 2.34.

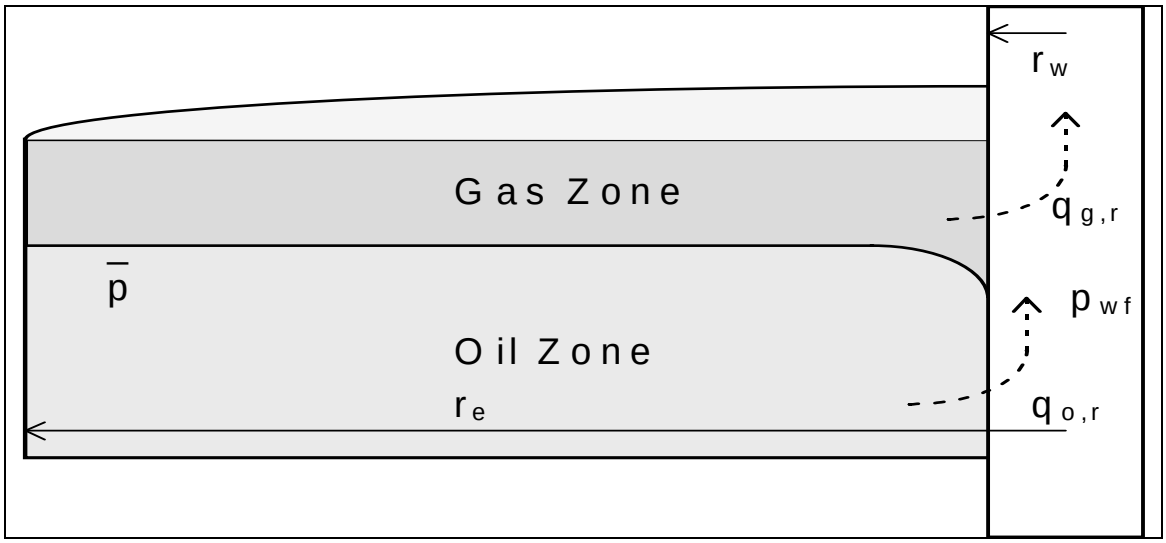

Figure 2.3: Conceptual Diagram of Reservoir

Equation 2.34 had to be further modified for use in this project. This is because the simulator is compositional, so mass flow rates are of more use than volumetric flow rates. In this report, mass rates are generally expressed in lb. mole per day.

For any time step the goal is to find a mass flow rate and composition that produce the same p_{wf} as the wellbore model, which is described in greater detail in another section. Hence, the process begins with a set of reservoir conditions, and a target mass flow rate.

1) Begin with an average reservoir pressure, $\bar{p}$, a total reservoir composition Z_r, and a target mass flow rate N. Also establish an initial estimate of p_{wf}.
2) Perform flash calculations to determine fluid properties and phase compositions for mixture Z at p_{wf}.

3) Based on these sandface phase properties, determine S_o and S_g. From these values, determine k_{ro} and k_{rg}.

4) Determine q_g and q_o. Use the properties from Step 2 to determine mass flow rates for each phase.

5) Now see if the process has converged.
   - If this mass flow rate is sufficiently close to the target rate, combine the oil and gas mass rates and compositions to determine the produced phase composition. Stop here.
   - If the mass flow rate is *not* sufficiently close to the target rate, adjust the p_{wf} via a Newton-Raphson process. Go back to Step 2.

2.3.1.1 Determining S_o, S_g, k_{ro}, & k_{rg}

In order to calculate relative permeabilities, a value for each phase saturation is required. Saturation values depend on the specific volume of each phase and liquid mole fraction, L.

$$S_o = \frac{M_o L / \rho_o}{M_o L / \rho_o + M_g (1-L) / \rho_g} \tag{2.35}$$

$$S_g = 1 - S_o \tag{2.36}$$

When these values are known, the relative permeability values come from a standard analytical form (Amyx, Bass, and Whiting, 1960).

$$k_{ro} = \left(\frac{(S_o - S_{or})}{(1 - S_{or} - S_{gr})} \right)^{n_{oil}} \tag{2.37}$$

$$k_{rg} = \left(\frac{(S_g - S_{gr})}{(1 - S_{or} - S_{gr})} \right)^{n_{gas}} \tag{2.38}$$

2.3.1.2 Determining Q_o & Q_g, and the Reservoir Mass Flow Rate

As mentioned before, Equation 2.34 must be modified to be appropriate for a compositional model. The volumetric flow rate equations follow:

$$q_o = \frac{0.00708 k k_{ro} h}{\mu_o}\left[\frac{\bar{p} - p_{wf}}{\ln(r_e/r_w) - 0.75}\right] \tag{2.39}$$

$$q_g = \frac{0.00708 k k_{rg} h}{\mu_g}\left[\frac{\bar{p} - p_{wf}}{\ln(r_e/r_w) - 0.75}\right] \tag{2.40}$$

The mass flow rates from the reservoir are as follows:

$$N_o = 5.615 q_o \rho_o / M_o \tag{2.41}$$

$$N_g = 5.615 q_g \rho_g / M_g \tag{2.42}$$

When the phase mass rates have been determined, the total mass rate for a given value of p_{wf} can be determined:

$$N = N_o + N_g \tag{2.43}$$

2.3.1.3 Determining the Produced Composition

When the method has converged to the target mass flow rate, the produced composition, Z_p, must be determined by combining the sandface phases, weighted by mass production rates. For each component i, the produced molar fraction can be found with this equation:

$$Z_{p,i} = \frac{N_o X_i + N_g Y_i}{N} \tag{2.44}$$

2.3.2 Making Time Steps

Making time steps requires an iterative process. Each time step begins with a given average reservoir pressure, $\bar{p}_k$, and ends at a lower average reservoir pressure, $\bar{p}_{k+1}$. However, the rates produced during the step depend on the average reservoir pressure *over the time step*, $\bar{p}_{k+½}$. Thus the following procedure is necessary.

1) Estimate $\bar{p}_{k+½}$. If $k=0$, say $\bar{p}_{k+½}=\bar{p}_k$. Otherwise, say $\bar{p}_{k+½}=\bar{p}_k+½\left(\bar{p}_k-\bar{p}_{k-1}\right)$.
2) Find the equilibrium mass flow rate and composition, M and Z_p (see above flow rate section and model overview section).
3) Determine the reservoir mass and composition at $k+1$. Use this to calculate $\bar{p}_{k+1}$.
4) Set $\bar{p}_{k+½}=½\left(\bar{p}_k+\bar{p}_{k+1}\right)$. If $\bar{p}_{k+½}$ changes significantly, convergence has not been reached - return to Step 2.

When this procedure has converged, the time step is considered complete. At this point, flow rates and important pressures can be recorded.

2.3.2.1 Determining the Reservoir Mass and Composition

The reservoir mass and composition determine the reservoir pressure and influence flow rates. Thus, it is necessary to keep track of the mass and composition. In this project, the basis for tracking was the number of pound moles of each component, N_i. After time step k, the rate of production is N, and the produced fluid composition is Z_p. Hence the resultant reservoir mass is determined as follows:

$$N_{k+1,i}=N_{k,i}-NZ_{p,i}\Delta t \tag{2.45}$$

The new total mass in the reservoir is given by Equation 2.46.

$$N_{k+1}=N_k-N\Delta t \tag{2.46}$$

The new reservoir composition can be described as follows:

$$Z_{r,i} = N_{k+1,i} \,/\, N_{k+1} \tag{2.47}$$

2.3.2.2 Determining the New Reservoir Pressure

Given the new reservoir composition, the reservoir volume, and temperature, it is relatively simple to determine the new reservoir pressure. While it is possible to find pressure directly from volume and temperature for a given EOS, it requires reformulating Equation 2.20. Instead, an iterative procedure is used. Using the molecular weight of the reservoir composition, the average reservoir density can be determined.

$$\rho_{res} = \frac{N_k M_{res}}{Ah\phi} \tag{2.48}$$

Now the reservoir fluid is flashed at different pressures until the reservoir density is matched.

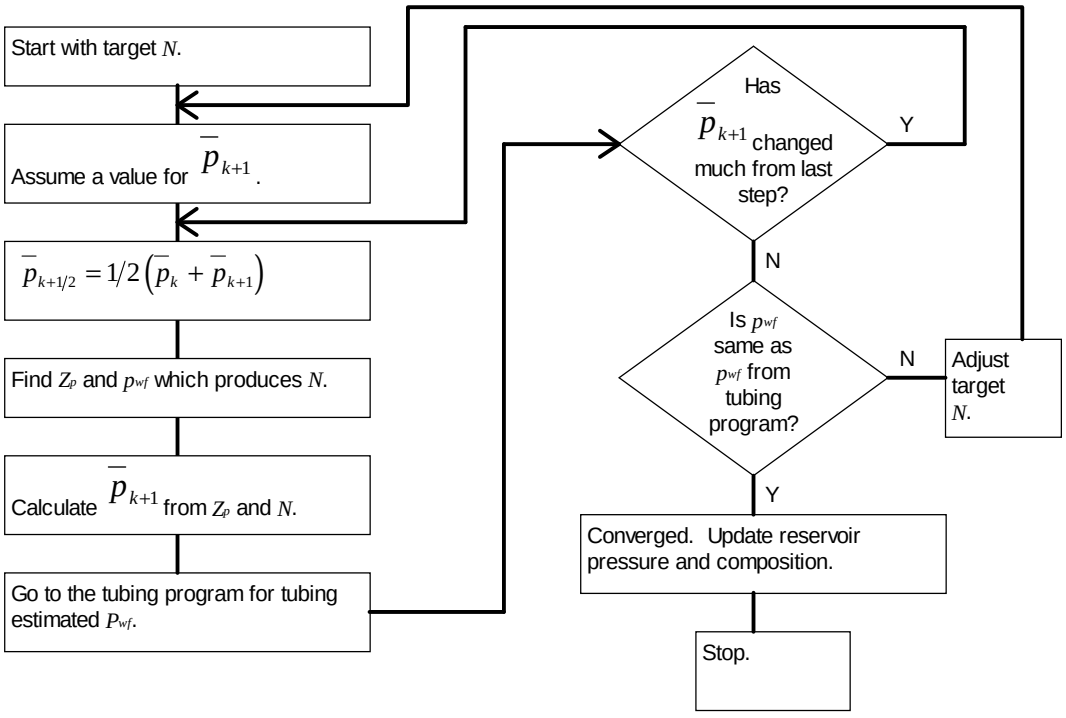

Figure 2.4 Reservoir Component Overview *(omits separator loop)*

2.4 Wellbore Model

In order to realistically model the behavior of a well producing oil and gas, a multiphase flow model is necessary. For this project, the model must be capable of handling a wide variety of mixtures at varying rates. In order to achieve this, the multiphase flow model of Aziz, Govier, and Fogarasi (1972) is used. For each step on the pressure traverse down the wellbore, the Aziz, Govier, and Fogarasi (AGF) method provides the flow regime, liquid holdup, and frictional pressure losses. For a given compositional flow rate beginning at a given wellhead pressure (pressure upstream of the choke) and temperature, the pressure at a given depth using the following technique is determined.

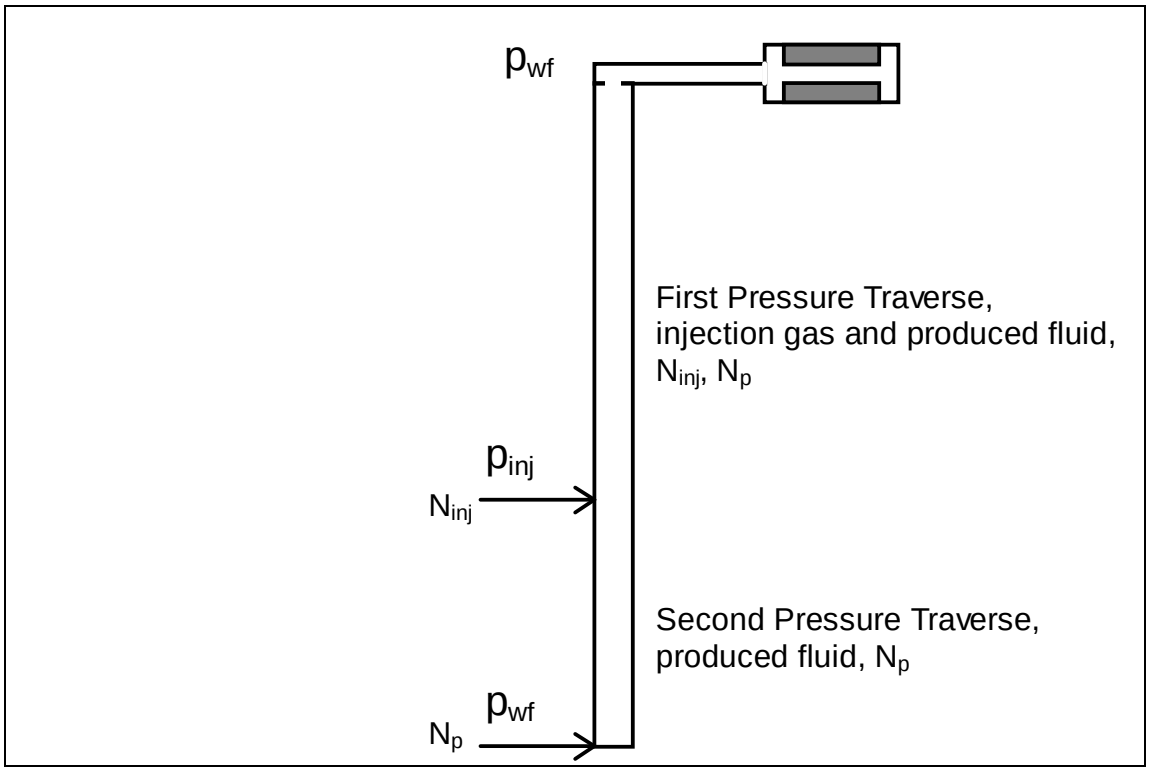

Figure 2.5 Pressure Traverse Diagram

1) Based on the pressure p_L, at depth L, assume a downstream pressure at a given change of depth, ΔL. To simplify, temperature is considered to vary linearly from surface to total depth. The initial guess for $p_{L+\Delta L}$ can be p_L.

2) Find the pressure at $p_{L+\frac{1}{2}\Delta L} = \frac{1}{2}\left(p_L + p_{L+\Delta L}\right)$.

3) Flash the flowing mixture at $p_{L+\frac{1}{2}\Delta L}$ and $T_{L+\frac{1}{2}\Delta L}$ to calculate the no-slip properties and compositions of the phases in this step of the traverse.

4) Use the multiphase flow correlation (AGF in this case) to determine the flow regime, liquid holdup, frictional pressure loss, and hydrostatic head over this pressure step.

5) Use the output of the multiphase flow correlation, and p_L to determine $p_{L+\Delta L}$.
   - If $p_{L+\Delta L}$ has changed, this step has not converged. Return to Step 2.
   - If it has not changed significantly, this step in the pressure traverse has converged.

6) Set p_L to $p_{L+\Delta L}$. Assume now that $p_{L+\Delta L} = p_L + (p_L - p_{L-\Delta L})$, and return to Step 2.

This process is repeated until the ultimate depth of interest is reached. For this project, it is repeated twice for each determination of bottomhole pressure. First, it is used to determine pressure at the point of gas injection, based on the surface pressure and the mixture of produced fluid and lift gas. Then this technique is used to determine the sandface pressure based on the injection-point pressure and produced fluid rate and composition.

2.4.1 Aziz, Govier, and Fogarasi (AGF) Multiphase Flow Correlation

For flow in a vertical pipe, it is common practice to consider two components of pressure drop along a flow path. The first component is hydrostatic head. The second is frictional loss. Frictional loss occurs both between the fluid and the pipe, and between the phases. Accurate, theoretical single phase flow models are available, but multiphase flow requires some sort of empirical correlation. Aziz, Govier, and Fogarasi (1972) developed such a correlation.

The difficulty in modeling multiphase flow lies in physical arrangement of the phases. The flow rates and properties of the phases determine the physical arrangement of the fluid, and this arrangement determines the hydrostatic head and frictional pressure loss. The AGF model provides for four flow regimes. Each regime represents a type of physical arrangement of the phases. The four regimes are bubble flow, slug flow, froth flow, and annular mist flow. These regimes are illustrated in Figure 2.6.

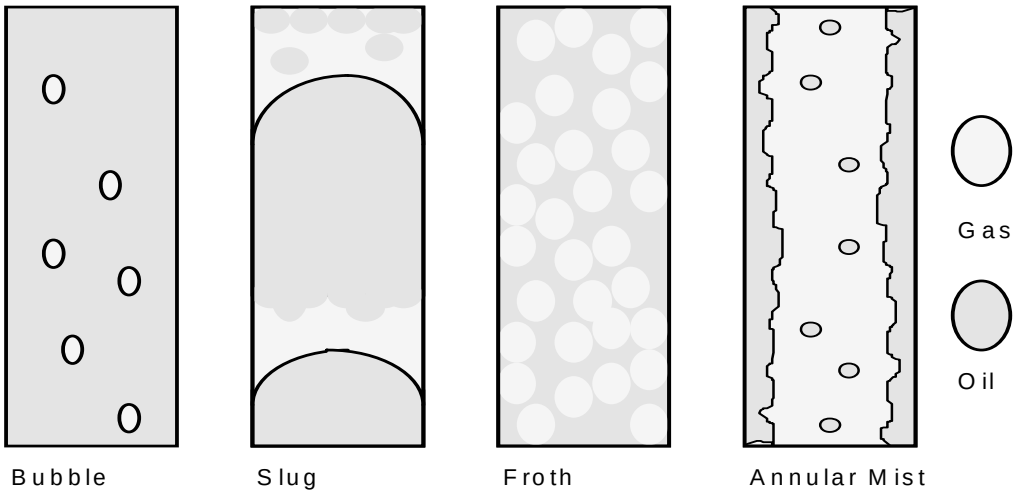

Figure 2.6: Illustration of Flow Regimes

In multiphase flow, the less dense phases will tend, due to gravity, to rise faster than the more dense phases. For fixed flow rates of each phase, this requires the denser phase to have a greater cross sectional area to compensate for the lower velocity. This leads to the holdup phenomenon. When the volumetric flow rate of each phase is known, there is an expected, stationary, in-situ volume fraction of liquid, C_L.

$$C_L = \frac{q_L}{q_L + q_G} \tag{2.49}$$

However, due of liquid holdup, the actual in-situ volume fraction of liquid, E_L, is not necessarily equal to C_L. For multiphase flow up a vertical pipe, $E_L \geq C_L$.

$$E_L = \frac{Volume_L}{Volume_L + Volume_G} \tag{2.50}$$

For each flow regime, the AGF method predicts a value of E_L based on C_L. The first step of the AGF method is determining the flow regime. To do this, first determine the superficial (no-slip, or assuming no holdup) velocity of each phase.

$$V_{sL} = \frac{q_L}{A} \tag{2.51}$$

$$V_{sG} = \frac{q_G}{A} \tag{2.52}$$

Another useful quantity is the mixture velocity.

$$V_M = V_{sL} + V_{sG} \tag{2.53}$$

From there, the modified superficial velocities are determined.

$$V_{MsL} = V_{sL}\left(\frac{\rho_L \sigma_{WA}}{\rho_{water}\sigma}\right)^{1/4} \tag{2.54}$$

$$V_{MsG} = V_{sG}\left(\frac{\rho_G}{\rho_{air}}\right)^{1/3}\left(\frac{\rho_L \sigma_{WA}}{\rho_{water}\sigma}\right)^{1/4} \tag{2.55}$$

Figure 2.7 illustrates the distribution of flow regimes based on these modified superficial velocities. The relationships are also described in Table 2.1.

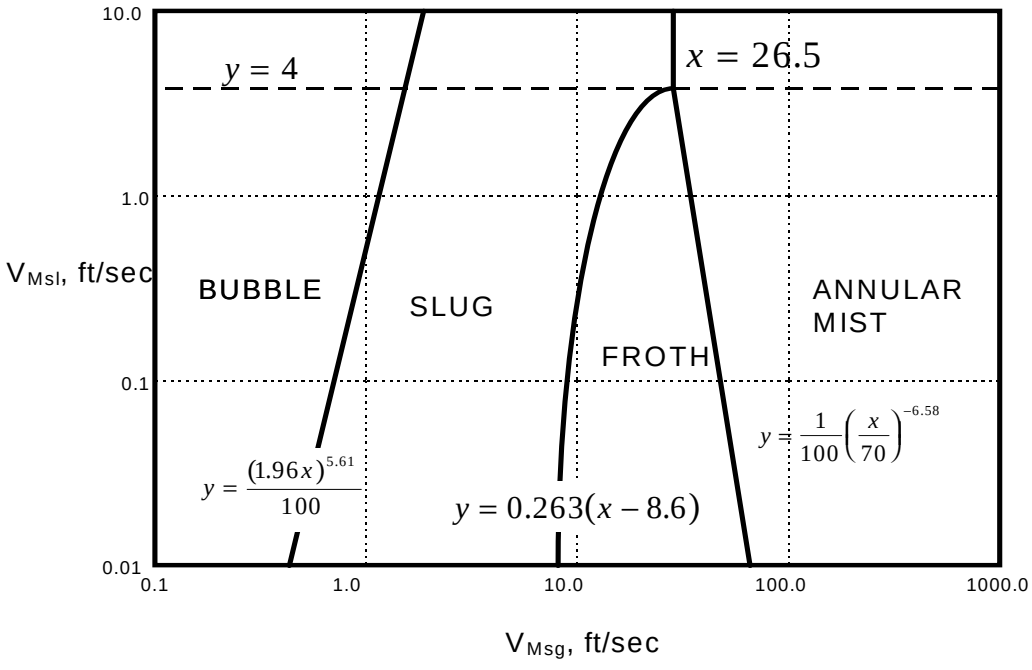

Figure 2.7: Flow Regime Map

Table 2.1: Description of Flow Regime Map

<table>
<tr><td colspan="4">$V_{MsL} > 4$</td></tr>
<tr><td>$V_{MsG} < \frac{(100y)^{.17211}}{1.96}$</td><td colspan="2">$V_{MsG} \geq \frac{(100y)^{.17211}}{1.96}$
$V_{MsG} < 26.5$</td><td>$V_{MsG} \geq 26.5$</td></tr>
<tr><td>Bubble</td><td colspan="2">Slug</td><td>Annular Mist</td></tr>
<tr><td colspan="4">$V_{MsL} \leq 4$</td></tr>
<tr><td>$V_{MsG} < \frac{(100y)^{.17211}}{1.96}$</td><td>$V_{MsG} \geq \frac{(100y)^{.17211}}{1.96}$
$V_{MsG} < \frac{y}{0.263} + 8.6$</td><td>$V_{MsG} \geq \frac{y}{0.263} + 8.6$
$V_{MsG} < 70\left((100y)^{-0.152}\right)$</td><td>$V_{MsG} \geq 70\left((100y)^{-0.152}\right)$</td></tr>
<tr><td>Bubble</td><td>Slug</td><td>Froth</td><td>Annular Mist</td></tr>
</table>

2.4.1.1 Bubble Flow Regime

This flow regime is characterized by a continuous liquid phase with the gaseous phase dispersed throughout in small bubbles. These bubbles rise through the liquid column. The AGF method requires the calculation of the rise velocity of bubbles in a stagnant liquid.

$$V_b = 1.41\left(\frac{g\sigma(\rho_L - \rho_G)}{\rho_L^2}\right)^{1/4} \tag{2.56}$$

Next determine the total gas rise velocity in the flowing fluid.

$$V_t = 1.2V_M + V_b \tag{2.57}$$

In the AGF technique, the in-situ liquid fraction is determined from Equation 2.58.

$$E_L = 1 - \frac{V_{sG}}{V_t} \tag{2.58}$$

From E_L, the hydrostatic pressure drop can be determined.

$$\Delta P_H = \Delta L\left(\left.\frac{dP}{dL}\right|_H\right) = \Delta L\left(\frac{g}{g_c}\left(\rho_L E_L + (1 - E_L)\rho_G\right)\right) \tag{2.59}$$

For the frictional pressure loss, first determine the Fanning friction factor. The Reynold's number can be used, to solve for the fanning friction factor, f_f, through an iterative solution of the Colebrook equation (Colebrook and White, 1937).

$$R_e = \frac{DV_M\rho_L}{\mu_L} \tag{2.60}$$

$$\frac{1}{\sqrt{4f_f}} = 1.74 - 2\log\left(\frac{2\varepsilon}{D} + \frac{18.7}{R_e\sqrt{4f_f}}\right) \tag{2.61}$$

This friction factor is used to solve for the frictional pressure drop with Equation 2.62.

$$\Delta P_F = \frac{2f_f V_M^2 \rho_L \Delta L}{g_c D} \tag{2.62}$$

2.4.1.2 Slug Flow Regime

This regime is characterized by alternating slugs of liquid and long bubbles of gas. The liquid is considered to be the continuous phase, and the liquid slugs can contain dispersed gas phase bubbles. The AGF technique treats slug flow in a fashion very similar to that for the bubble flow regime.

$$V_b = 0.345\left(\frac{Dg(\rho_L - \rho_G)}{\rho_L}\right)^{1/2} \tag{2.63}$$

Next determine the rise velocity in the flowing fluid with Equation 2.57. From E_L, the hydrostatic pressure drop can be determined with Equation 2.59.

The friction factor is determined exactly as it would be for the bubble flow regime. Then the frictional pressure loss can be determined with Equation 2.64.

$$\Delta P_F = \frac{2 f_f V_M^2 \rho_L \Delta L(E_L)}{g_c D} \tag{2.64}$$

2.4.1.3 Annular Mist Flow Regime

In this flow regime, gas velocities are high enough to result in a continuous gaseous phase. The liquid phase is distributed in an annular ring around the perimeter of the pipe, and as dispersed droplets carried within the gaseous core. The AGF correlation borrows the treatment of Duns and Ros (1963) for this regime. The important assumption of this treatment is that frictional forces cause the liquid phase to travel at the same velocity as the gaseous phase.

$$E_L = C_L = \frac{V_{sL}}{V_M} \tag{2.65}$$

From E_L, the hydrostatic pressure drop can be calculated with Equation 2.59. AGF uses only gas flow to determine frictional losses.

$$R_e = \frac{D V_{sG} \rho_G}{\mu_G} \tag{2.66}$$

R_e is used to determine the friction factor from Equation 2.61. Equation 2.67 provides the frictional loss.

$$\Delta P_F = \frac{2 f_f V_{sG}^2 \rho_G \Delta L}{g_c D} \tag{2.67}$$

2.4.1.4 Froth Flow Regime

The froth flow regime represents a transition between the slug and annular mist flow regimes. Pressure drops are resolved by linear interpolation between the two flow regimes.

2.4.2 Overview

Figure 2.8 illustrates the complete pressure traverse algorithm.

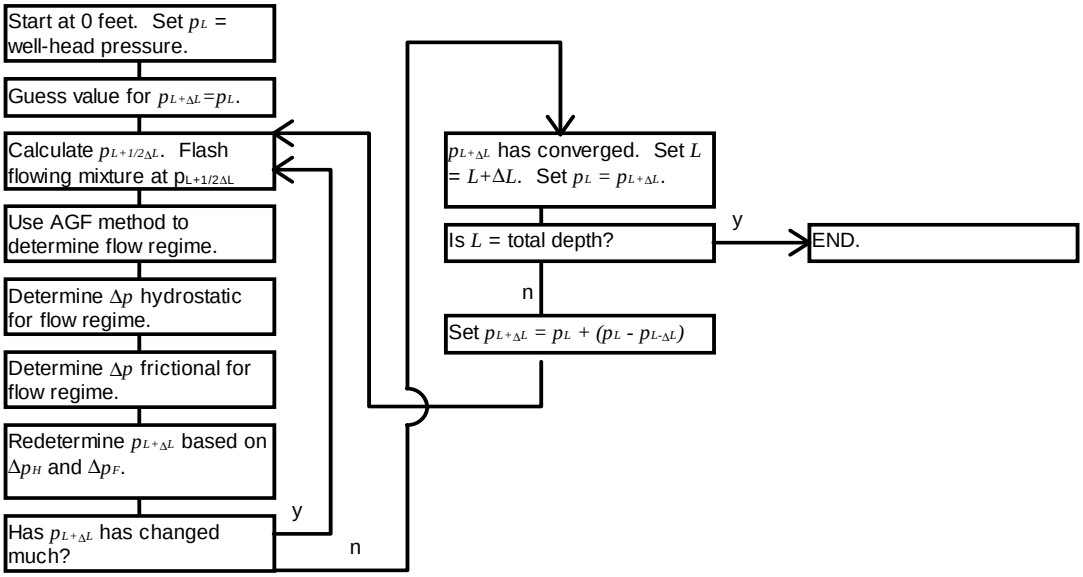

Figure 2.8: Flowchart For Pressure Traverse

2.5 Choke Model

Surface chokes are flow restrictions placed between the top of the tubing and the separator. In theory, the choke should be small enough to cause critical flow. In critical flow, the fluid flow rate across the choke is dependent only on the upstream pressure, and is independent of the downstream pressure. This has many advantages. Separator pressure can be changed, within reason, without altering the wellhead or

sandface pressures. Additionally, if there are multiple wells on a manifold, high pressure wells will not kill, or even inject into lower pressured wells. Critical flow occurs when the ratio of downstream to upstream pressures, y, falls below a critical value, usually near 0.5. Good practice calls for chokes to be small enough to cause critical flow, but large enough to avoid killing the well. In reality, many wells are not in critical flow (Fortunati, 1972). Figure 2.9 illustrates the relationship between mass flow rate and critical flow.

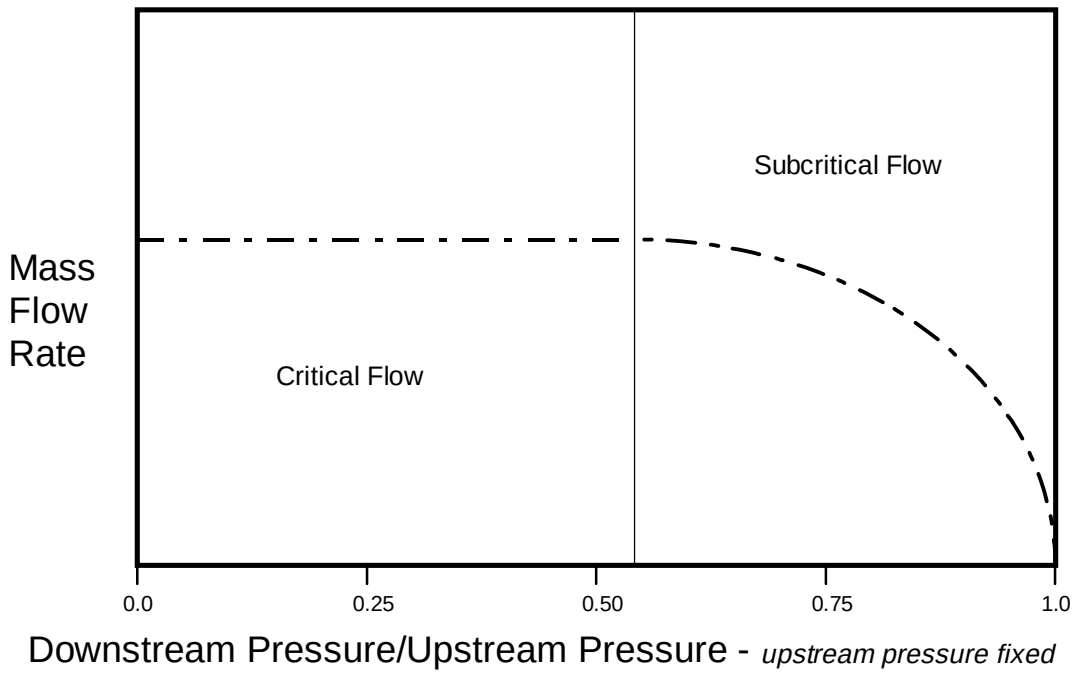

Figure 2.9: Conceptual Illustration of Critical and Subcritical Flow

Many models have been developed to describe choke flow. Many of these, however, only describe single-phase flow. Additionally, most of the multiphase flow models specifically assume critical flow. For this project the choke was modeled as proposed by Sachdeva, Schmidt, Brill, and Blais (1986).

This model is sophisticated enough to cope with both multiphase flow and subcritical flow. Before describing the model in more detail, it is necessary to describe its limitations:

- Flow is one dimensional;

- Phase velocities are equal at the throat;
- The predominant pressure term is accelerational;
- The quality is constant for high speed processes;
- The liquid phase is incompressible.

2.5.1 Critical/Subcritical Boundary

The first step in the Sachdeva et. al. model is to determine the boundary between critical flow and subcritical flow. The ratio of downstream pressure to upstream pressure at this boundary is y_c. This critical ratio is found at the root of Equation 2.68.

$$F(y) = y - \left\{ \frac{\frac{k}{k-1} + \frac{(1-x_1)V_L(1-y)}{x_1 V_{G1}}}{\frac{k}{k-1} + \frac{n}{2} + \frac{n(1-x_1)V_L}{x_1 V_{G2}} + \frac{n}{2}\left[\frac{(1-x_1)V_L}{x_1 V_{G2}}\right]^2} \right\}^{\frac{k}{k-1}} = 0 \qquad (2.68)$$

The root of this equation can be determined through an iterative process. Figure 2.10 illustrates the form of this equation.

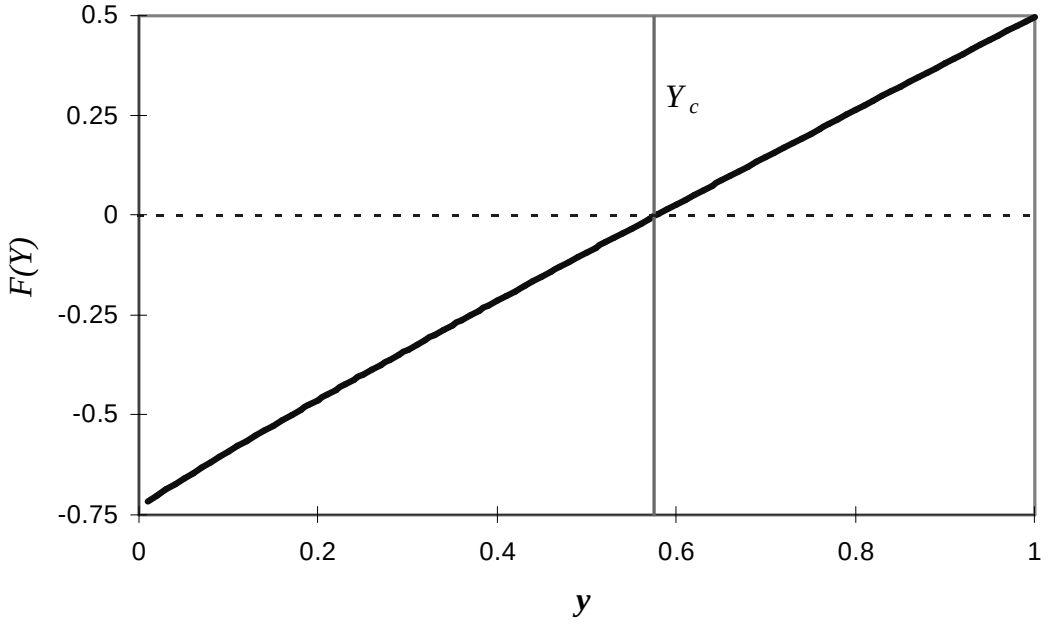

Figure 2.10: Form Of *F(y)*

2.5.2 Mass Rates

Once the critical ratio has been determined, proceed to find the mass rate across the choke with the following equations:

$$y = \frac{P_2}{P_1} \tag{2.69}$$

$$y_u = \max(yc, y) \tag{2.70}$$

$$V_{G2} = V_{G1} y^{-\frac{1}{k}} \tag{2.71}$$

$$\rho_{m2} = \left(x_1 V_{G1} y^{-\frac{1}{k}} + (1 - x_1) V_L \right)^{-1} \tag{2.72}$$

$$M = A_c C_D \sqrt{\left\{ 2 g_c \cdot 144 P_1 \rho_{m2}^2 \left[\frac{(1 - x_1)(1 - y_u)}{\rho_L} + \frac{x_1 k}{k - 1} \left(V_{G1} - y V_{G2} \right) \right] \right\}} \tag{2.73}$$

2.5.3 Liquid Flow

One of the limitations with the Sachdeva, et. al. choke model is that it cannot handle single-phase liquid flow. Fortunately, there are good single-phase liquid flow models. For single phase-liquid flow, assume that the pressure drop through the choke is equal to the kinetic energy pressure drop divided by the square of a drag coefficient (Economides, Hill, and Ehlig-Economides, 1994).

$$q = 22{,}800 C (D_2)^2 \sqrt{\frac{\Delta P}{\rho}} \tag{2.74}$$

C is a flow coefficient of the choke, based on the choke diameter and Reynold's number. This coefficient ranges from 0.92 to 1.2.

2.5.4 Choke Solution Technique

In this project, the downstream pressure, flow rate, and flow composition were considered to be given. The unknown is the upstream pressure that can move the given mass rate and composition across the choke, against the given downstream pressure. Figure 2.11 is an example of the resultant function. For a given rate, the correct upstream pressure is found iteratively.

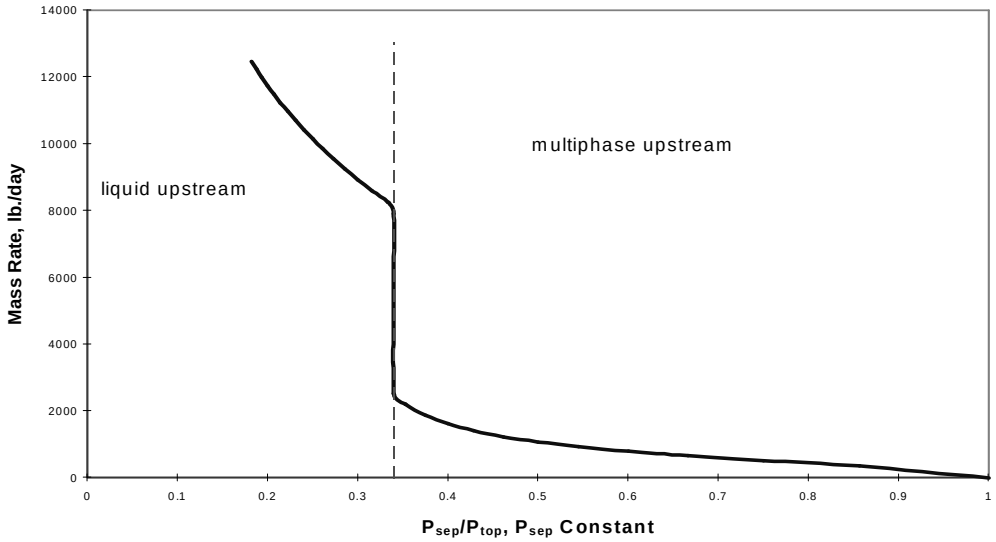

Figure 2.11: Mass Rate as a Function of Upstream Pressure

2.6 Separator Model

A separator is a vessel which takes a multiphase fluid stream, and separates the stream into two streams, each predominated by one phase. Generally, oil or gas delivered for sale must meet certain restrictions on content, meaning that the oil should not be too gassy, and the gas should have limited fluid drop-out. Separators are controlled by adjusting the internal pressure. The amount of each output stream depends upon the separator pressure. If the separator pressure is too high, the resultant oil will contain too much gas, which is likely to flash off in the stock-tank. If the pressure is too low, a great deal of medium-weight molecules (pentane, etc.) may flash into the gas phase, reducing the volume of the more valuable oil phase.

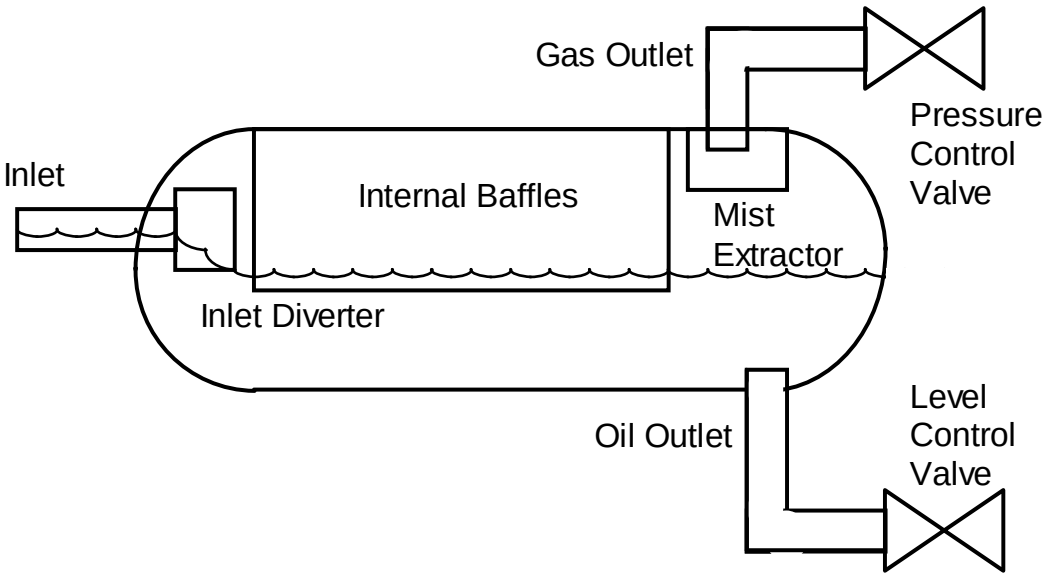

Figure 2.12: Separator Diagram, after *Fundamentals of Petroleum (1981)*

In the process of separation, a mixture of hydrocarbons is moved from wellhead pressure to stock tank pressure. Typically, it is economically desirable to reach stock tank conditions with as large a volume of liquid stream as is possible. The best way of doing so is to closely emulate a differential liberation process (McCain, 1990). If the fluid passed directly from the wellhead to stock tank, the pressure would suddenly drop a great deal, and the process would approximate a "flash" liberation process. A differential liberation process is one where pressure is lowered in a stepwise fashion. After each pressure reduction, the resultant gas phase is removed. It can be shown that, in terms of maintaining liquid mass content, differential liberation is much more efficient than flash liberation.

In order to emulate differential liberation, several separators are connected in series. Each separator has progressively lower pressure. The result is more efficient separation. For this project, a series of three separators are used. Figure 2.13 illustrates the design of the separator system.

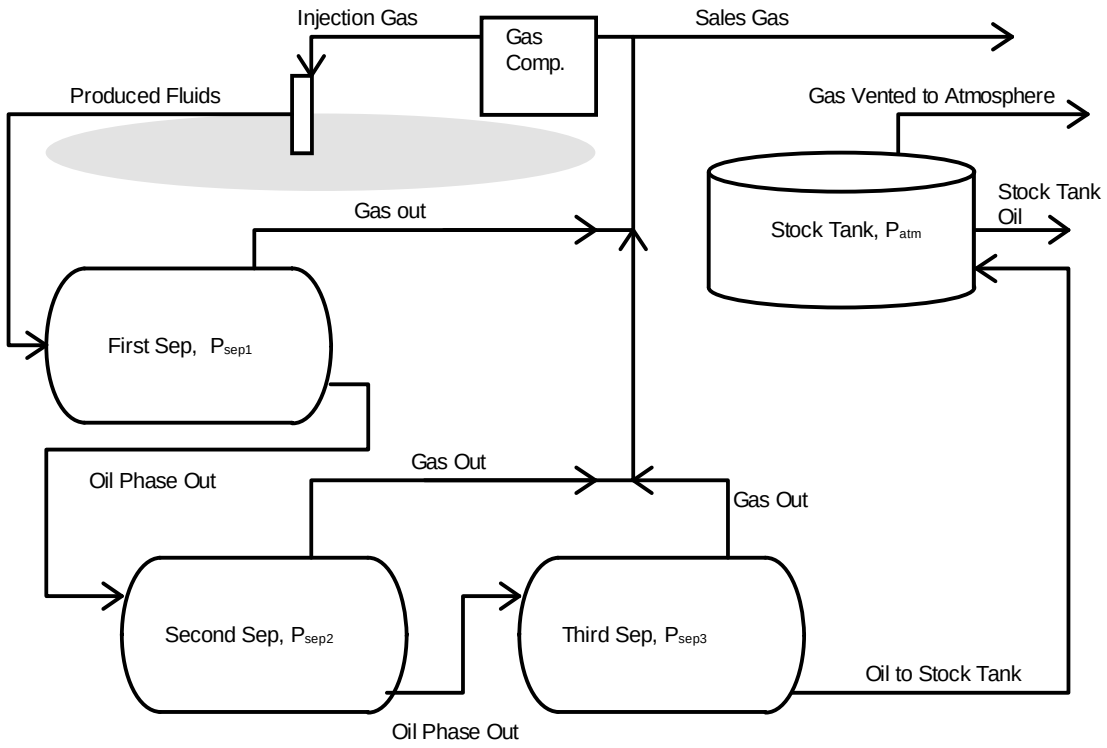

Figure 2.13: Separator System Design

The scheme for separation works as follows:

- Begin with a mixture of the produced fluid and the lift gas. This mixture has a mass flow rate of N_{wh}, and a composition of Z.
- Perform a flash calculation of Z at p_{sep1}. The liquid mole fraction at this stage is L_1. The gas phase is Y_1, with mass rate $N_{wh}(1-L_1)$. The oil phase is $N_{wh}L_1$ with composition X_1.
- Flash X_1 at p_{sep2}. The liquid mole fraction at this stage is L_2. The gas phase is $N_{wh}L_1(1-L_2)$ with composition Y_2. The oil phase is $N_{wh}L_1L_2$ with composition X_2.
- Flash X_2 at p_{sep3}. The liquid mole fraction at this stage is L_3. The gas phase is $N_{wh}L_1L_2(1-L_3)$ with composition Y_3. The oil phase is $N_{wh}L_1L_2L_3$ with composition X_3.
- Flash X_3 at p_{atm}. The liquid mole fraction in the stock tank is L_{st}. The sales oil stream is $N_{wh}L_1L_2L_3L_{st}$ with composition X_{st}. The sales mass rate (lb./day) can be figured by multiplying the mass rate (lbmol/day) by the molecular weight of X_{st}. The volume rate can be figured by dividing this mass rate by the fluid's density.
- The gas stream can be determined by combining the three separator gas streams. The mass rate (lbmol/day) is $N_{wh}(1-L_1)+ N_{wh}L_1(1-L_2)+ N_{wh}L_1L_2(1-L_3)$.

- Notice that gas from the stock tank is lost to the atmosphere.

The gas from this process is divided between gas to be sold and lift gas. The oil produced from the stock tank is sold.

2.7 Economics

Any optimization requires an objective function. Petroleum engineering economics provide many possible objective functions for project analysis. These include; rate of return, payout, net present value, return on investment. For this project, the net present value criteria was selected. Net present value provides the *discounted* value of a future cash flow (Thompson and Wright).

Table 2.2 is an example calculation of net present value for a project with initial cost of $60,000 and the net cash flow presented in the table. This example is provided by Thompson and Wright (1985). Notice that each year, the discounting factor is compounded into the previous year's factor.

Table 2.2: Net Present Value Illustration from Thompson and Wright (1985)

Year	Net Cash Flow	Present Value Discounted at 10%
0	-60,000	-60,000
1	37,100	33,727
2	16,800	13,884
3	12,200	9,166
4	8,640	5,901
5	5,440	3,378
6	250	141
	Net Present Value for Project	$ 6,198

In this project, each simulated flow stream is converted into a discounted present value. Each time step includes charges for operating expense, separation cost, and a cost for the horse-power needed to compress the lift-gas. The horsepower needed for compression is estimated with Equation 2.75 (Economides et al., 1994).

$$HHP = 2.23 \times 10^{-4} q_g \left[\left(\frac{P_{surf}}{P_{in}} \right)^{0.2} - 1 \right] \tag{2.75}$$

Obviously, the net present value also depends on a wide variety of factors, such as the discount factor and assorted charges.

2.8 Overview and Flowchart

Describing these models separately is somewhat difficult because they must be employed together in an iterative fashion. For example, the separator model determines the composition of the lift gas, but that composition is determined by the fluid which flows into the separator model. The composition entering the separator is largely determined by the composition of the produced fluid. The composition of the produced fluid, in turn, depends on lift gas performance, which is determined largely by the composition of the gas! Additionally, the composition of the fluid entering the separator is dependent on the lift-gas composition. Hence, there are several circular loops. There are other examples of such loops in the model, but it will be clearer to demonstrate with a flow chart, Figure 2.14.

Each time step in a simulation with the combined model requires the following steps:

1) Begin at time t, lift composition Z_L, and reservoir pressure $\overline{p}_t$. Assume that lift gas composition for the time step Δt is Z_L. Assume a value for $\overline{p}_{t+\Delta t}$. Assume a mass flow rate.
2) Find $\overline{p}_{t+\frac{1}{2}\Delta t}$ from $\overline{p}_t$ and $\overline{p}_{t+\Delta t}$.
3) Find the reservoir sandface pressure, p_{wfr}, for the rate. Also find the produced fluid composition.

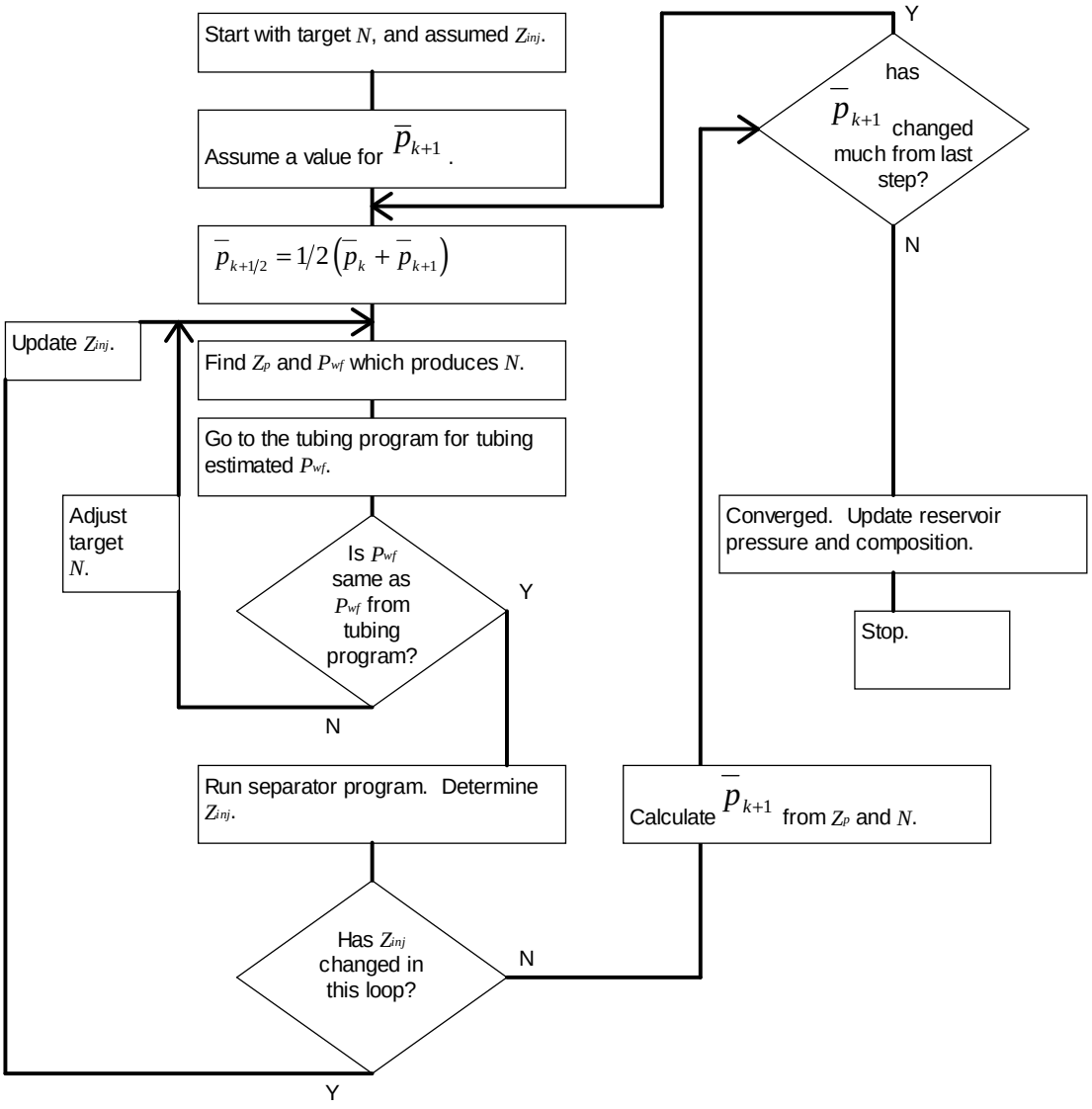

Figure 2.14: Flowchart of Complete Model

4) Find the wellhead pressure which allows the produced fluid and lift-gas to flow across the choke against the first separator pressure.
5) Use the pressure traverse with the combined fluid composition and mass flow rate to find the pressure at the point of injection.
6) Use the pressure traverse with the produced fluid composition and mass flow rate to find the tubing pressure at the sandface, p_{wft}.
7) If the difference between p_{wfr} and p_{wft} is too large, choose another mass flow rate and go back to Step 3.
8) Combine the produced fluid and lift gas, Z_L, and use the separator model to determine the new lift gas composition. If the lift gas composition has not changed

significantly, proceed. If the lift gas composition has changed significantly, return to Step 3 with the new Z_L.

9) Find the resultant $\overline{p}_{t+\Delta t}$ from producing the reservoir at this rate and composition for Δt. If $\overline{p}_{t+\Delta t}$ has changed significantly, return to Step 2. Otherwise, the time step has concluded. All quantities can be updated, and the next time step can begin.

2.9 Difficulties in Implementation

Implementing this model has entailed overcoming many difficulties. This is largely due to the connectivity of the various models. Most of the difficulties have regarded convergence and root finding. The best example is the process of finding the mass flow rate which produces the same sand-face pressure for both the reservoir, and the tubing.

This is a difficult problem to address, because unlike single phase flow cases, there may be more than one possible stable equilibrium state. Figure 2.15 illustrates this point. It is possible to have a stable single phase liquid rate, and two different multiphase rates. Of the multiphase rates, only one is stable. The Newton-Raphson technique is used to find the equilibrium. It is possible to find as many as three equilibria, depending on the starting rate. To solve this problem, various types of logical statements are used to select an appropriate solution, and the direction of change during the Newton-Raphson step.

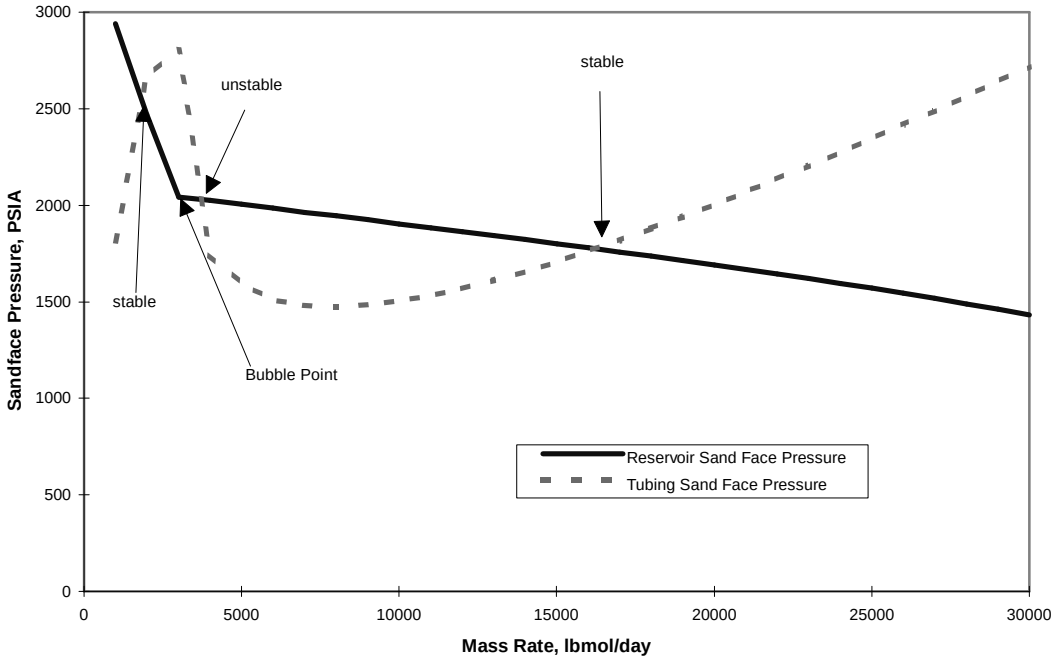

Figure 2.15: Multiphase Inflow Performance Curves

3. Nonlinear, Multivariate Optimization

In general, many engineering projects will fit the description of an optimization problem. There will generally be some objective function that the engineer wishes to maximize or minimize, such as fuel efficiency, or velocity, or rate of transmission. The variables that the engineer can change to maximize or minimize the objective function are called decision variables. The family of mathematical, or heuristic techniques used to find minima or maxima is referred to as optimization.

In many cases, there is only one decision variable. This type of optimization is illustrated in Figure 3.1. While these sorts of problems can be very difficult to solve, the level of difficulty increases geometrically as additional decision variables are added. Optimization problems involving more than one decision variables are referred to as multivariate optimization. Additional difficulties arise when the function calls become computationally expensive, or when the function becomes nondifferentiable.

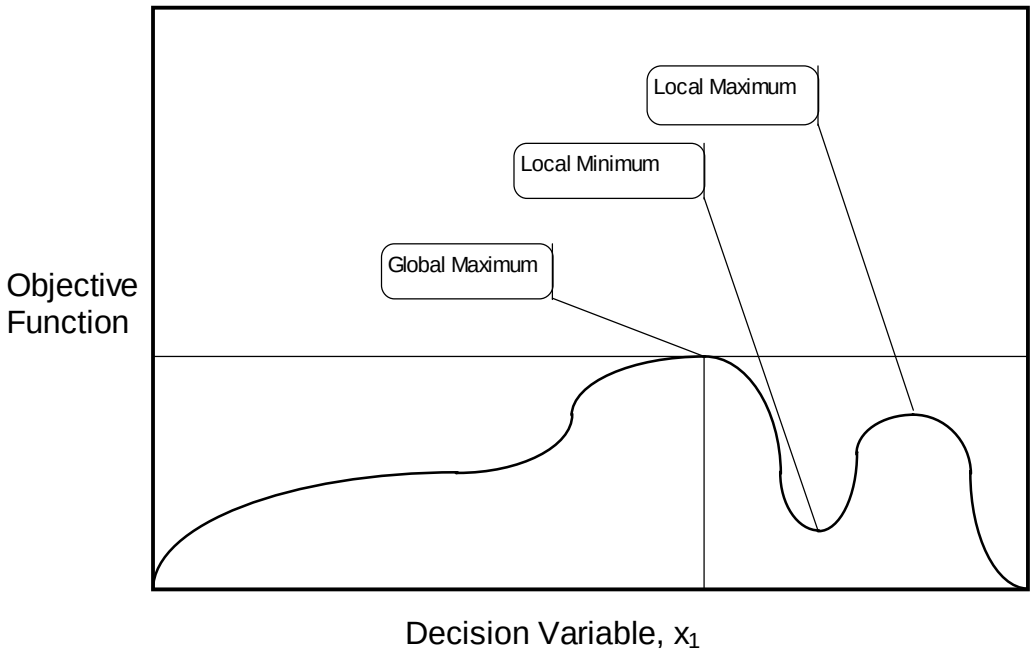

Figure 3.1: Univariate Optimization Problem

This project addressed three families of optimization techniques. These families include; Newton type techniques, polytope techniques, and genetic algorithms. Each of these families has advantages and disadvantages, and it may not be clear at the beginning of a project which technique will work best. In part, the purpose of this project was to determine which techniques work best for this type of petroleum engineering optimization problem.

3.1 Newton Type Techniques

These techniques are the classic tools used to solve multivariate optimization techniques. The standard technique is the Newton method itself. Following the description of the Newton method, different types of modifications to Newton's method will be described.

Before describing these techniques, it is necessary to explain that it is conventional to assume that optimization refers to the *minimization* of an objective function. This being the case, maximization requires minimizing the negative of the objective function. This is illustrated in Figure 3.2.

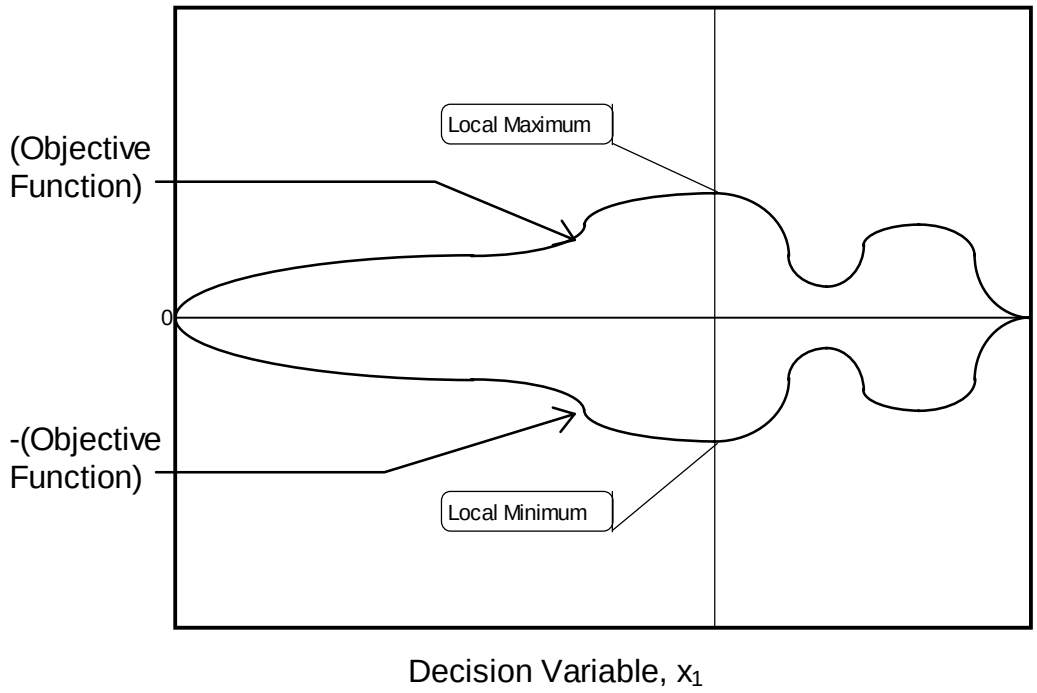

Figure 3.2: Treating Maximization as Minimization

3.1.1 Newton's Method

Newton's method is a technique whereby the local curvature of the objective function is used to approximate a quadratic function (a bowl in three dimensions). The next position in the optimization process is the bottom of the approximate quadratic function. Theoretically, the minimum is reached where the gradient is zero. After each move, the quadratic is solved again for another step.

Start with a vector of n decision variables $\tilde{x}$. The objective function is then $F(\tilde{x})$. Newton's method creates a quadratic approximation of F by using the first three terms of the Taylor expansion of F. Using a step, $\tilde{p}$, towards the minimum.

$$F(\tilde{x}_k + \tilde{p}) = F(\tilde{x}_k) + \tilde{g}_k^T \tilde{p} + \frac{1}{2}\tilde{p}^T G_k \tilde{p} + O_3 \tag{3.1}$$

$$F(\tilde{x}_k + \tilde{p}) - F(\tilde{x}_k) \approx \tilde{g}_k^T \tilde{p} + \frac{1}{2}\tilde{p}^T G_k \tilde{p} \tag{3.2}$$

$$\tilde{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \tag{3.3}$$

Where, $\tilde{p}$ is the step length to reach $\tilde{x}_{k+1}$,

$$\tilde{p} = \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \vdots \\ \Delta x_n \end{bmatrix} \tag{3.4}$$

Also, $\tilde{g}$ is the gradient vector of the surface at $\tilde{x}$,

$$\tilde{g} = \begin{bmatrix} \frac{\partial F}{\partial x_1} \\ \frac{\partial F}{\partial x_2} \\ \vdots \\ \frac{\partial F}{\partial x_n} \end{bmatrix} \tag{3.5}$$

And G is the Hessian matrix of F.

$$G = \begin{bmatrix} \frac{\partial^2 F}{\partial x_1 \partial x_1} & \frac{\partial^2 F}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 F}{\partial x_1 \partial x_n} \\ \frac{\partial^2 F}{\partial x_2 \partial x_1} & \frac{\partial^2 F}{\partial x_2 \partial x_2} & \cdots & \frac{\partial^2 F}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 F}{\partial x_n \partial x_1} & \frac{\partial^2 F}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 F}{\partial x_n \partial x_n} \end{bmatrix} \tag{3.6}$$

Function F in Equation 3.2 is rearranged to form the quadratic function, Q.

$$Q(\tilde{x}_k + \tilde{p}) = F(\tilde{x}_k) + \tilde{g}_k^T \tilde{p} + \frac{1}{2} \tilde{p}^T G_k \tilde{p} \tag{3.7}$$

We wish to minimize Q, so we must differentiate with respect to $\tilde{p}$.

$$\frac{\partial}{\partial} \frac{Q(\tilde{x}_k + \tilde{p})}{\tilde{p}} = \tilde{g}_k^T + G_k \tilde{p} \tag{3.8}$$

The minimum must occur when the derivative goes to zero, so the solution must fit Equation 3.9.

$$\tilde{g}_k^T = -G_k \tilde{p} \tag{3.9}$$

Equation 3.9 can be rearranged into Equation 3.10.

$$\tilde{p} = -G_k^{-1} \tilde{g}_k^T \tag{3.10}$$

After evaluating F, $\tilde{g}$, and G at $\tilde{x}_k$, use Equation 3.10 to determine $\tilde{p}$. Given that $\tilde{x}_{k+1} = \tilde{x}_k + p$, it is easy to find the next position in the optimization process. Eventually, the process reaches a point where convergence criteria are met. Typically, the process is discontinued when all elements of either $\tilde{p}$ or $\tilde{g}$ become sufficiently small in magnitude.

Newton's method usually converges very quickly. The convergence behavior is quadratic in nature (Gill, Murray, and Wright, 1981). If the initial starting point, $\tilde{x}_0$, is sufficiently close to the minimum, convergence is guaranteed. However, as the minimum is typically unknown, this is of limited value.

One clear limitation of this technique is that it involves the calculation of numerous first and second derivatives. In cases where function evaluations are computationally expensive, and analytical derivatives are not available, each iteration of this technique can be very expensive. For n decision variables, $\frac{n^2 + n}{2}$ second derivatives are required, even given that $\frac{\partial^2 F}{\partial x_i \partial x_j} = \frac{\partial^2 F}{\partial x_j \partial x_i}$.

Another limitation comes from the fact that the solution of $\tilde{p}$ may not be a minimum. The quadratic function Q may have a stationary point at $\tilde{x}_{k+1} + \tilde{p}$, but it is not necessarily a minimum. This point may be a maximum, or even a saddle point. To be sure that $\tilde{x}_k + \tilde{p}$ is a minimum, G must be positive definite. That means that the determinant of G must be positive, and all diagonal elements must be positive as well. If the Hessian matrix is not positive definite, the step $\tilde{p}$ may still be convergent; however, if this is not the case, the step $\tilde{p}$ may be divergent. The possible outcomes are illustrated in Figure 3.3.

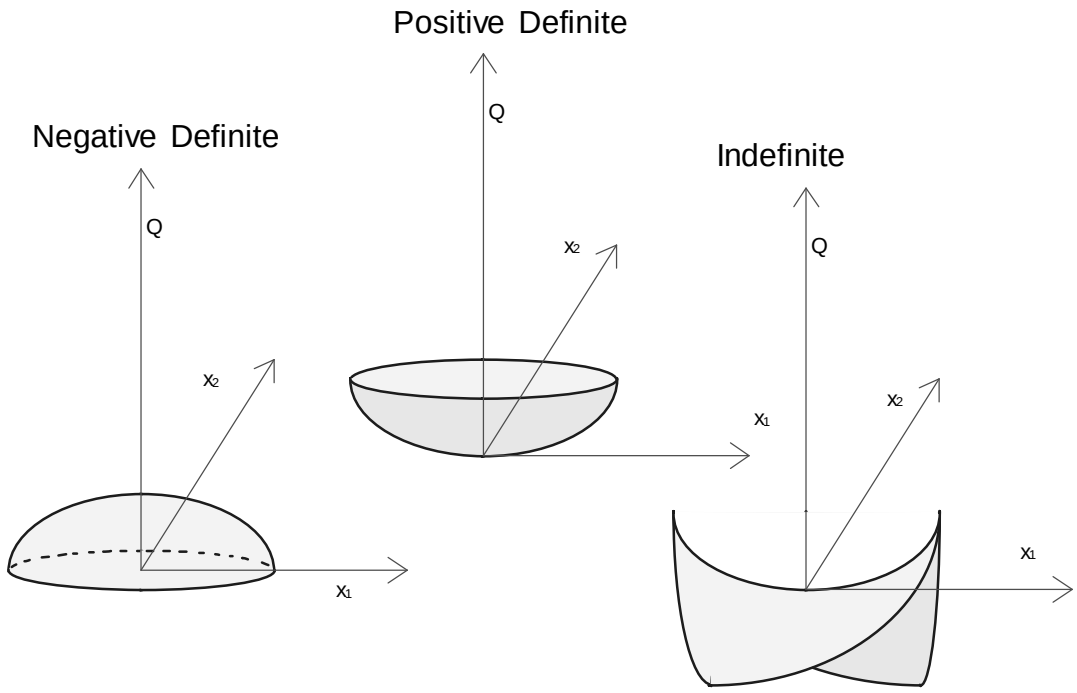

Figure 3.3: Illustration of Possible Quadratic Shapes

3.1.2 Method of Steepest Descent

Because the convergence of Newton's method depends upon whether the Hessian matrix is positive definite, many alternatives have been devised which assure a positive definite Hessian matrix. The simplest of these techniques is the method of steepest descent. In this method, the Hessian matrix is replaced by the identity matrix multiplied by a scalar.

$$cI \rightarrow G \tag{3.11}$$

While this method exhibits tremendous stability, and always results in a downhill step, it is extremely slow. In any sort of an "ax-blade problem", i.e. an elongated valley, the method will "hemstitch" across the blade until it finally converges. This technique is rarely appropriate.

3.1.3 Marquardt Modification

Rather than simply replacing the Hessian matrix, G, with the identity matrix, the Marquardt modification alters the Hessian by adding a positive constant to each element along the diagonal (Gill, Murray, and Wright, 1981).

$$G_M = G + \mu I \tag{3.12}$$

$$\tilde{p} = G_M^{-1}\tilde{g} \tag{3.13}$$

Provided that the constant is sufficiently large, the Hessian matrix will be positive definite, and a descent step is assured. This method is a simple way to replace negative eigenvalues with positive eigenvalues. Instead of calculating eigenvalues, the Marquardt method relies on a type of trial and error approach to finding the correct constant to add. The algorithm described below was the technique used in this project.

1. At the beginning of a step, the constant, μ, is very small. Evaluate G and $\tilde{g}$ at $\tilde{x}$.
2. The step, $\tilde{p}$, is computed.
3. The dot product of $\tilde{p}$ and the gradient, $\tilde{g}$, indicates whether the direction is actually in descent or ascent.
4. Check if the direction of $\tilde{p}$ is in descent or not.
    a) If the direction is in descent, the step is accepted, and the objective function is reevaluated at $\tilde{x} + \tilde{p}$. Check to see if the procedure has converged.
        i) If the method has converged, stop here.
        ii) If not, replace $\tilde{x}$ with $\tilde{x} + \tilde{p}$, return to Step 1.
    b) If, however, the direction is actually ascending, increase μ by an order of magnitude. Return to Step 2.

The Marquardt method tends to be very robust because it acts like the method of steepest descent in indefinite or negative definite regions, and like Newton's method in positive definite regions.

3.1.4 Quasi-Newton Techniques

It has already been noted that a limitation of Newton's method is the expensive calculation of the elements of the Hessian matrix. Quasi-Newton methods are techniques whereby the Hessian matrix in current iterations has been constructed from prior data and the current gradient. A Taylor series expansion of the gradient function about $\tilde{x}$ leads to Equation 3.14.

$$\tilde{p} = G^*_{\tilde{x}+\tilde{p}}\left(\tilde{g}_{\tilde{x}+\tilde{p}} - \tilde{g}_{\tilde{x}}\right) \tag{3.14}$$

There are an infinite number of solutions for $G^*_{\tilde{x}+p}$, so an approximation is necessary. A useful form is the Broyden, Fletcher, Goldfarb, Shanno method (Gill, et al., 1981).

$$G^*_{\tilde{x}+\tilde{p}} = G^*_{\tilde{x}} + \frac{\left(\tilde{g}_{\tilde{x}+\tilde{p}} - \tilde{g}_{\tilde{x}}\right)\left(\tilde{g}_{\tilde{x}+\tilde{p}} - \tilde{g}_{\tilde{x}}\right)^T}{\left(\tilde{g}_{\tilde{x}+\tilde{p}} - \tilde{g}_{\tilde{x}}\right)^T \tilde{p}} - \frac{G^*_{\tilde{x}}\tilde{p}\tilde{p}^T G^*_{\tilde{x}}}{\tilde{p}^T G^*_{\tilde{x}}\tilde{p}} \tag{3.15}$$

The first Newton step uses the identity matrix, and thereafter, a Hessian is constructed. Quasi-Newton techniques require less function calls to construct their Hessian matrixes, but do not converge as quickly as Newton's method.

3.1.5 Line Search Procedures

One of the best refinements to Newton's method is to institute a line search. This procedure can be added to Newton's method, or to any of the modifications of Newton's method. The concept of the line search technique is to reduce the dimensions of the optimization problem at each iteration. After the step of Newton's method has been determined, the objective function will be optimized along the line $\tilde{x} + \rho\tilde{p}$. The line search is implemented to find a value of ρ which minimizes $F(\tilde{x} + \rho\tilde{p})$. The value of ρ returned by the Newton method is 1.0, and this may not be minimal, even along the line $\tilde{x} + \rho\tilde{p}$.

There are two philosophies for using line searches. The first is to conduct a thorough search to find the true global minimal along the line $\tilde{x} + \rho\tilde{p}$. The other philosophy,

used in this project, is to only perform a cursory search along $\tilde{x} + \rho\tilde{p}$. The logic of this second approach is that there is no guarantee that the true global minimum of $F(\tilde{x})$ for n decision variables lies along $\tilde{x} + \rho\tilde{p}$, so devoting a lot of function calls to finding the minimum along $\tilde{x} + \rho\tilde{p}$ is probably not worthwhile.

The cursory approach used in this project considers two possibilities. The objective function is evaluated at $\tilde{x} + 1\tilde{p}$. If $F(\tilde{x} + 1\tilde{p})$ is less than $F(\tilde{x})$, extrapolation to a larger value of ρ is attempted. However, If $F(\tilde{x} + 1\tilde{p})$is greater than $F(\tilde{x})$, interpolation to a smaller value of ρ is attempted. The number of function calls during the line search is minimized. The algorithm is as follows (Bard, 1974).

1. Find the greatest possible value of ρ that does not violate any constraints of decision variables. Call this value α.
2. Choose a value $\rho_n = \min(0.95\alpha, 1)$.
3. Evaluate $F(\tilde{x} + \rho_n\tilde{p})$.
4. If $F(\tilde{x} + \rho_n\tilde{p}) < F(\tilde{x})$ then:
   a) Choose a value $\rho_{n+1} = \min\left(2\rho_n, \rho_n + \frac{3}{4}(\alpha - \rho_n)\right)$.
   b) Evaluate $F(\tilde{x} + \rho_{n+1}\tilde{p})$.
   c) If $F(\tilde{x} + \rho_{n+1}\tilde{p}) < F(\tilde{x} + \rho_n\tilde{p})$, accept ρ_{n+1} as the correct step length. Terminate line search.
   d) If $F(\tilde{x} + \rho_{n+1}\tilde{p}) \geq F(\tilde{x} + \rho_n\tilde{p})$, accept ρ_n as the correct step length. Terminate line search.
5. If $F(\tilde{x} + \rho_n\tilde{p}) \geq F(\tilde{x})$ then:
   a) Use cubic approximation to find minimum from $\tilde{x}$ to $\tilde{x} + \rho_n\tilde{p}$.
   b) $\gamma = \tilde{g}^T\tilde{p}$
   c) $\rho_{n+1} = -\frac{\gamma\rho_n^2}{2\left[F(\tilde{x} + \rho_n\tilde{p}) - F(\tilde{x}) - \gamma\rho_n\right]}$
   d) Replace ρ_n with ρ_{n+1}. Evaluate $F(\tilde{x} + \rho_n\tilde{p})$.

e) If $F(\tilde{x} + \rho_n \tilde{p}) \geq F(\tilde{x})$ go back to Step 5.c. Otherwise, terminate the line search.

6. Terminate the search.

An obvious flaw with this technique is that if the Newton step is not in a direction of descent, this method can fail. However, if it is used with a method that guarantees a descent direction, Marquardt or steepest descent, this procedure can enhance convergence considerably.

Line search procedures are not guaranteed to provide good results. There may be circumstances where the line search produces undesirable results, such as convergence to local minima. On the whole, however, a line search procedure will enhance convergence.

3.2 Polytope Optimization Algorithm

Unlike Newton type optimization methods, the polytope algorithm methods does not require any gradient information. In fact, the polytope method was developed to cope with nonsmooth surfaces. The polytope constitutes a heuristic, common sense solution for optimization.

A polytope for n decision variables consists of $n+1$ points in a concave arrangement on the surface of the objective function. The polytope moves toward the minimum by flipping its worst point around the centroid of the other points. Aside from flipping, the polytope also expands, contracts, shrinks, and restarts. Each of these behaviors serves a purpose for the ultimate convergence of the polytope to the global minimum. Figure 3.4 illustrates some of these behaviors.

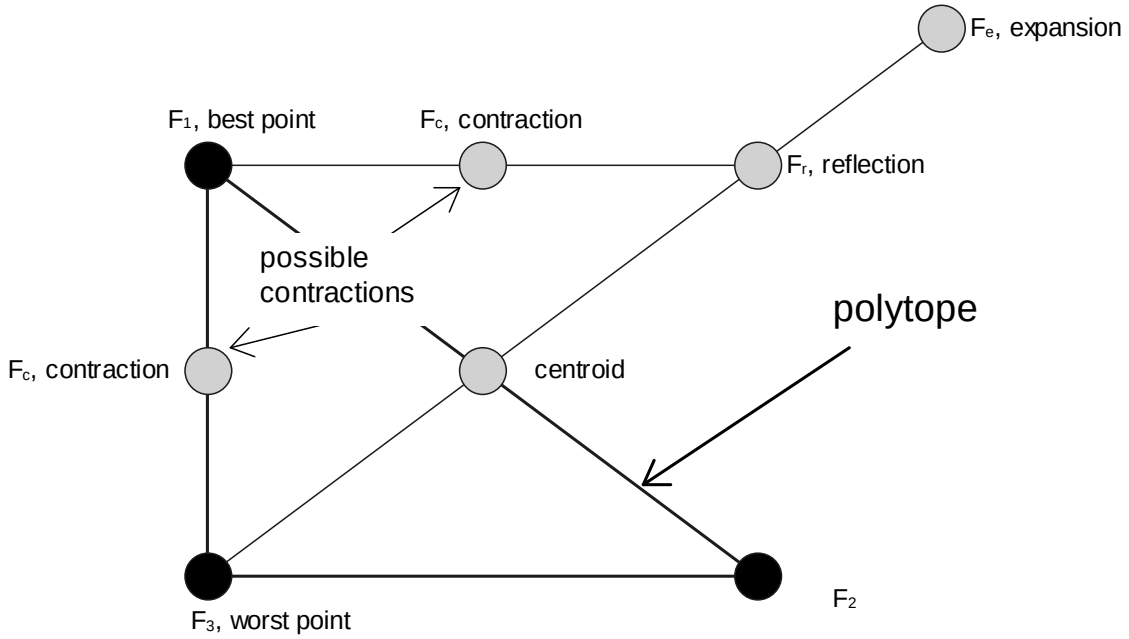

Figure 3.4: Illustration of Polytope Behavior

The polytope algorithm is as follows:

1. Choose an initial $n+1$ points $(\tilde{x}_1, \tilde{x}_2, \cdots; \tilde{x}_n, \tilde{x}_{n+1})$. Evaluate the objective function at each of the $n+1$ points $(F(\tilde{x}_1), F(\tilde{x}_2), \cdots; F(\tilde{x}_n), F(\tilde{x}_{n+1}))$.
2. Order the points from best (least) to worst (greatest).
3. Generate the centroid, $\tilde{c}$, of the first n points. $\tilde{c}_i = \frac{1}{n}\sum_{j=1}^{n} \tilde{x}_{j,i}$.
4. Determine a point of reflection, $\tilde{x}_r$, across the centroid from the worst point, $\tilde{x}_{n+1}$. $\tilde{x}_{r,i} = \tilde{c}_i + \alpha(\tilde{c}_i - \tilde{x}_{n+1,i})$.
5. Evaluate $F(\tilde{x}_r)$.
6. If $F(\tilde{x}_1) \le F(\tilde{x}_r) \le F(\tilde{x}_n)$, $\tilde{x}_r$ is neither a new best point nor a new worst point. Replace $\tilde{x}_{n+1}$ with $\tilde{x}_r$ and return to Step 2.
7. If $F(\tilde{x}_r) < F(\tilde{x}_1)$, $\tilde{x}_r$ is a new best point. Attempt expansion.
   a) Find the expansion point, $\tilde{x}_e$. $\tilde{x}_{e,i} = c_i + \beta(x_{r,i} - c_i)$.
   b) Evaluate $F(\tilde{x}_e)$.

c) If $F(\tilde{x}_e) < F(\tilde{x}_r)$ the expansion is successful. Replace $\tilde{x}_{n+1}$ with $\tilde{x}_e$ and return to Step 2.

d) If $F(\tilde{x}_e) \geq F(\tilde{x}_r)$ the expansion has failed. Replace $\tilde{x}_{n+1}$ with $\tilde{x}_r$ and return to Step 2.

8. If $F(\tilde{x}_r) > F(\tilde{x}_n)$, the polytope should be contracted. Contract the polytope.

a) Find the contraction point, $\tilde{x}_c$.

i) If $F(\tilde{x}_r) \geq F(\tilde{x}_{n+1})$, then $\tilde{x}_{c,i} = \tilde{x}_{1,i} + \gamma\left(x_{n+1,i} - \tilde{x}_{1,i}\right)$.

ii) If $F(\tilde{x}_r) < F(\tilde{x}_{n+1})$, then $\tilde{x}_{c,i} = \tilde{x}_{1,i} + \gamma\left(x_{r,i} - \tilde{x}_{1,i}\right)$.

b) Evaluate $F(\tilde{x}_c)$.

c) If $F(\tilde{x}_c) < \min\left(F(\tilde{x}_r), F(\tilde{x}_{n+1})\right)$. Replace $\tilde{x}_{n+1}$ with $\tilde{x}_c$ and return to Step 2.

d) If $F(\tilde{x}_c) \geq \min\left(F(\tilde{x}_r), F(\tilde{x}_{n+1})\right)$. Contract further, with a smaller value of γ. Return to 8.a.

The contraction in this algorithm is a modification suggested by Gill, Murray, and Wright (1981). This contraction is toward the best point, while the unmodified polytope contracts toward the centroid.

There are other polytope operations that need to be described. The first is shrinking. If the contraction step has failed twice, the polytope is shrunk. In shrinking, the best point is maintained, and all other points move closer to the best point. The polytope then begins climbing again.

Another operation is the restart. In the restart operation, the best point is maintained, and the remainder of the polytope is replaced by a new, regular polytope. This prevents the polytope from becoming stretched and contorted. Restarts are performed after $2n$ iterations without the best point changing. If the restarted polytope goes $2n$ iterations without the best point changing, the best point is considered to be the global minimum, and the algorithm halts.

The last polytope operation is to check for degeneracy. The polytope should not collapse into a shape with only $n-1$ dimensions. In other words, a polytope with two decision variables has three vertices. It becomes degenerate when all three vertices lay on a line. This can occur for any value of n greater than 1, and it prevents the polytope from exploring most of the desired search space. This problem may be encountered when the polytope reaches a constraint. If the polytope has degenerated, it can either be restarted, or a point can be displaced in a direction normal to the degenerated polytope.

3.3 Genetic Algorithms

Genetic algorithms are a method of optimization that draw an analogy to the process of natural selection (Goldberg, 1989). Variables are discretized, and converted into binary string referred to as members.

Each member of a generation has a chance to breed with another member of the population. The odds that a member is able to breed depends on its fitness, or objective function value. The next generation of members is composed of the "offspring" of the prior generation. Because the odds of a member being able to reproduce depends on that member's fitness, each progressive generation should be better than the next. In this fashion, the genetic algorithm should slowly converge towards the global optimum.

The method is particularly apt for many petroleum problems, because the variables are discrete. Many engineer-controlled petroleum engineering parameters are actually discrete, such as tubing diameter. For continuous variables, sufficient binary bits can be used to provide a reasonable approximation of continuity.

Before describing genetic algorithms further, it should be noted that genetic algorithms are purely heuristic, and have no solid mathematical origin. Genetic algorithms rely upon an analogy to a process, evolution, which appears to be an optimizing phenomenon. However, there is no proof that engineering systems should evolve in a fashion similar to biological entities. Furthermore, it is a fallacy to view evolution as an optimization process. Evolution is a process by which self-replication is assured, and,

is not the process aimed at some optimal design. In biological design, there are many examples of "suboptimal" design (Dawkins, 1986). Likewise, genetic algorithms, while very robust, can converge to suboptimal solutions.

Table 3.1: Illustration of Binary to Real Conversion

Decision Variable	**X_1**	**X_2**
Minimum Value	*1*	*1*
Maximum Value	*64*	*64*
Step Size	*1*	*1*
Number of Values	*64*	*64*
Number of Bits	*6*	*6*
Binary Value	010001	111110
Value	18	63
Binary Member	**010001111110**	
Decimal Member	**(18,63)**	

The first step of the genetic algorithm routine is to generate an initial first population. Each generation has a number of members equal to the total number of bits in each member. In this study the initial population is generated entirely randomly. The value of each bit in each member is randomly chosen from [0,1]. There are, however, more rigorous methods for generating the initial population, such as the Fang (1980) algorithm which distributes the initial members as evenly spaced as is possible.

Table 3.1 serves to illustrate the conversion of a binary string, or member, to a vector of decision variables. Each member of the population is converted into a decision variable vector, $\tilde{x}$, and the value of the objective function for each member, $F(\tilde{x})$,is evaluated.

When each member has been evaluated, the fitness of members must be ranked for reproduction. The ranking is performed by allotting each member space on a "roulette wheel". The size of each space depends on the members fitness. Breeding partners are drawn from the wheel at random, which usually prevents the weakest members from reproducing. An example of such a "roulette wheel" is illustrated by Table 3.2 and Figure 3.5.

Table 3.2: Fitness Table Construction

Member	String	Binary Values		Decimal Values		Objective Function Value	Fitness %
1	010001111110	010001	111110	18	63	81	37
2	001011010110	001011	010110	12	23	35	16
3	001010000101	001010	000101	11	6	17	8
4	110011100011	110011	100011	52	36	88	39
						Σ 221	100

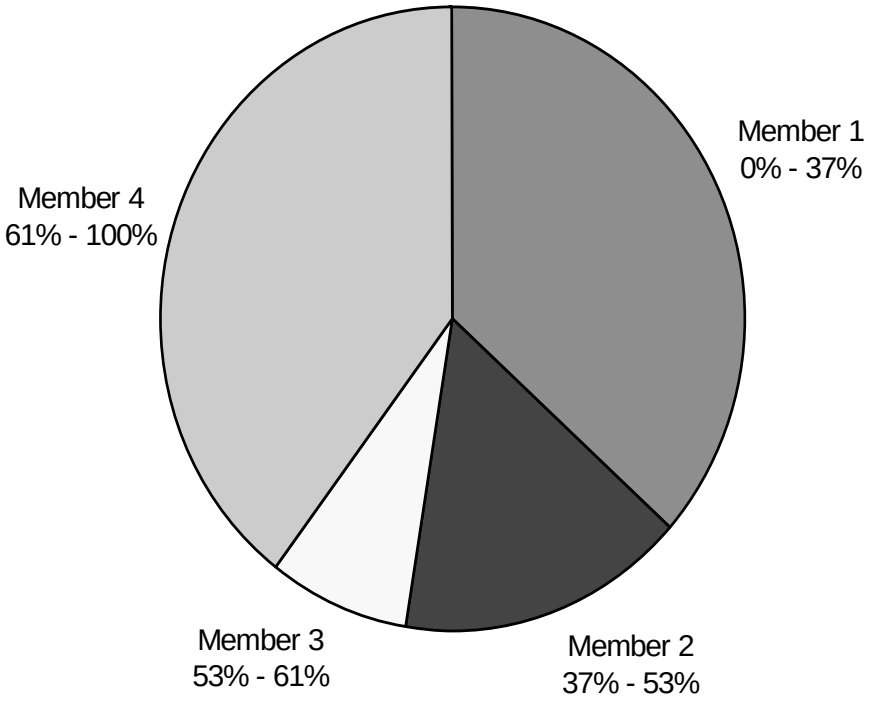

Figure 3.5: Genetic Algorithm Roulette Wheel

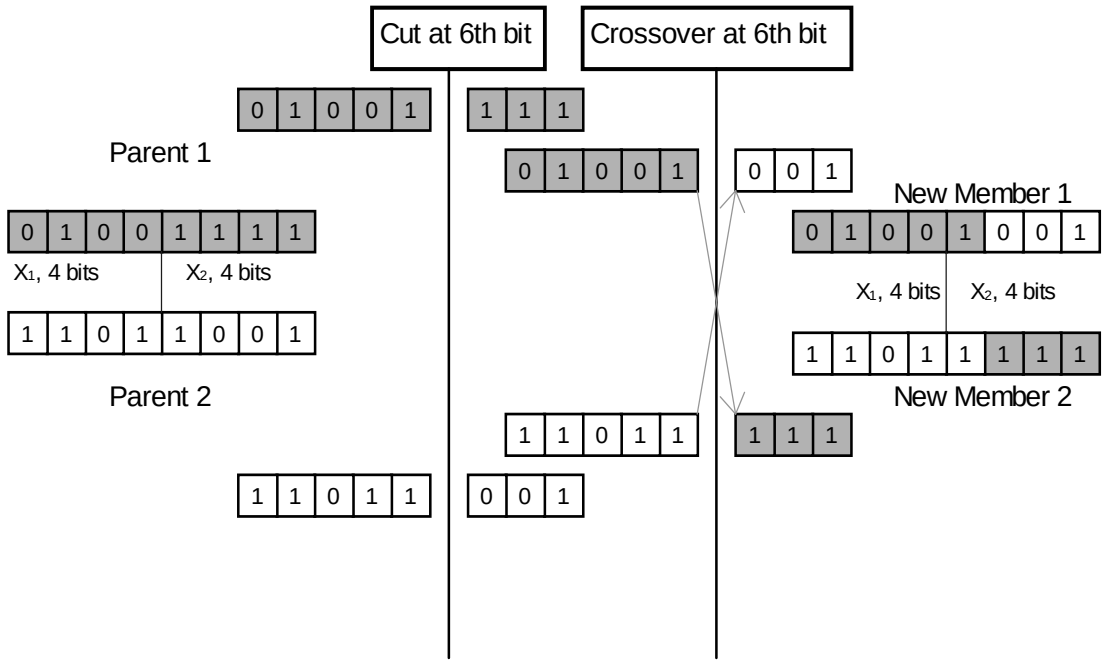

Figure 3.6: Crossover Illustrated

Table 3.3: Crossover Example

random numbers		parent members		parent strings		random bit	new member
1st	2nd	1st	2nd	1st	2nd		
0.52	0.04	2	1	001011010110	010001111110	4	001001111110
		1	2	010001111110	001011010110	4	010011010110
0.28	0.79	1	4	010001111110	110011100011	10	010001111011
		4	1	110011100011	010001111110	10	110011100110

When the breeding members have been selected, crossovers are performed. The crossover step involves choosing a random bit. One new member will have the "genetic material" of the first breeding partner before the selected bit, and the "genetic material" of the second breeding partner in and after the selected bit. The other new member from this mating will be constructed from the remaining genetic material, as is illustrated in Figure 3.6 and Table 3.3.

When the new members have been generated, the process of mutation is begun. There is a probability for each member that one or more bits, selected at random, will have their value switched, as illustrated in Figure 3.7. This measure is designed to introduce fresh material into the "gene pool". Without mutation, genetic algorithms are prone to prematurely converge upon suboptimal solutions (Fujii, 1993).

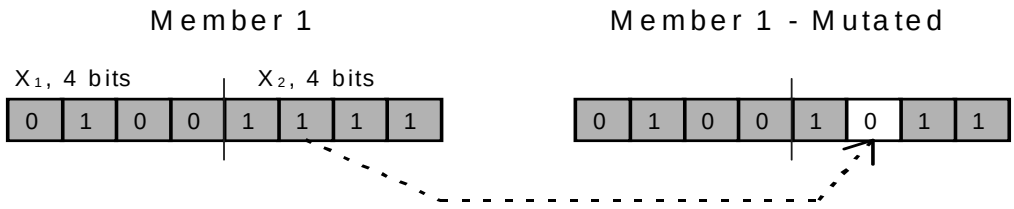

Figure 3.7: Mutation Illustrated

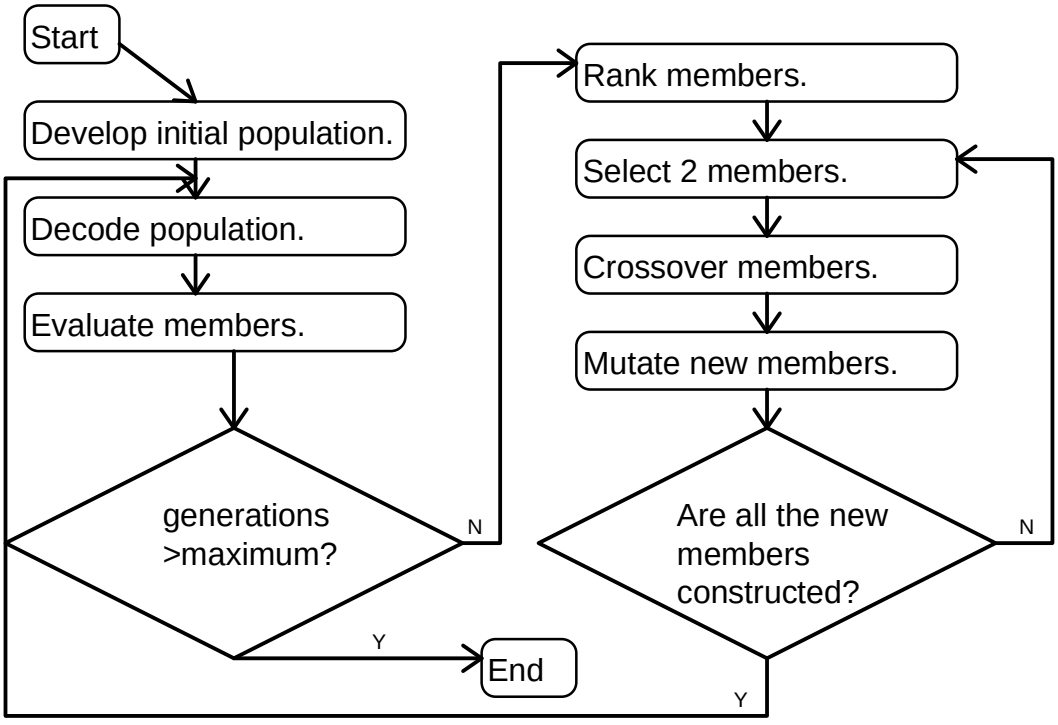

Figure 3.8: Genetic Algorithm Flowchart

The basic genetic algorithm is outlined in Figure 3.8. There are modifications of the basic algorithm that can significantly improve performance. The Fang (1980) algorithm is one example.

Another modification that can significantly improve speed is the addition of “memory”. Genetic algorithms can only produce a limited amount of possible points in the search space. Because “genetic material” is conserved at each mating, the algorithm is likely to repeat points that have already been evaluated. Thus, it is useful to store the results of previous runs, and look them up before reevaluating them.

This idea can be expanded to include windows surrounding points that have already been evaluated. In early generations, the genetic algorithm should explore as much space as is possible. So in early generations any new population member sufficiently close to a previously evaluated member takes on the value of the previously evaluated member. As the generations proceed, and the genetic algorithm begins to converge, the windows shrink, until they only include the point itself.

Another improvement is to always preserve the best point. It is possible with genetic algorithms for the best member’s string to be discarded, especially when high mutation rates are used. Preserving the best string from one generation to the next assures that the genetic algorithm does not diverge, even if it is not converging.

Fitness scaling is a technique which can prove to be useful. In some cases, methods other than the simple linear creation of the “roulette wheel” may produce better convergence. A comparison of some of these methods will be discussed in Section 4.

These and other modifications are explored in Section 4. A great deal of time was spent studying genetic algorithms because of their great potential for this type of problem. On the other hand, genetic algorithms do have limitations. Genetic algorithms tend to be robust, but they can be quite slow. Another problem with genetic algorithms is that there is no clear convergence criterion, excepting the passage of a given number of generations or function calls. The advantages include that genetic algorithms work well with nonsmooth surfaces, are capable of dealing with many variables, and easily deal with discrete variables.

4. Optimization Results

In Section 3, three families of optimization techniques are described. In this section, the effectiveness of these techniques with the model developed for this project is described. For several cases, the methods are compared and contrasted, and considerable attention is given to fine tuning genetic algorithm parameters. After each problem is presented, the efficiency of the techniques is discussed.

4.1 Problem 1

The first problem addressed has two decision variables. The reservoir is described in Table 4.1. The decision variables are the first separator pressure and the tubing diameter. There is no time dependence for these variables. All other operating parameters are also listed in Table 4.1.

Table 4.1: Problem 1 Description

Reservoir						
Areal Extent of Reservoir (acres)					160.	
Thickness (feet)					50.	
Porosity					0.30	
Initial Res. Pressure					3,500.00	
Reservoir Temperature (^{o}Rankine)					650.	
Permeability (md)					5.0	
S_{wc}, S_{or}, S_{gr}					0.0, 0.3, 0.0	
n_o, n_g					1.5,1.0	
Reservoir Depth					8,000	
Reservoir Fluid						
C_1	C_2	C_3	C_5	C_8	C_{10}	C_{11+}
0.05	0.13	0.12	0.13	0.19	0.19	0.19
Production Parameters (non-variable)						
injection rate (Mscf/day),					300.0	
choke diameter (1/64 inches),					30.0	
surface line diameter (inches),					1.0	
P_{sep2}/P_{sep1}					0.95	
P_{sep3}/P_{sep2}					0.90	

The surface of the objective function evaluated for this problem is illustrated in Figure 4.1. For this reservoir, with these conditions, an excessive tubing size prevents the well from flowing. Inspection of the surface also reveals that it is not very smooth. This is a good indicator that the gradient family of techniques are not likely to be effective.

For the genetic algorithm techniques, each solution was run five times with different random seeds, producing a family of curves to compare. Thus, for both the Newton type technique and the polytope technique, five random start locations were used.

The optimum solution for this problem is a tubing diameter of 2.69 inches combined with a first separator pressure of 164.1 psia. This combination corresponds to a NPV of $6,241,428. This combination was found with the polytope algorithm. The genetic algorithm could only sample the objective function at 4,096 discrete locations, and this was not one of them. Of all the locations sampled by the genetic algorithm, the optimum tubing diameter was determined to be 3.10 inches, and the optimum separator pressure was determined to be 155.56 psia. This combination corresponds to a NPV of $6,233,859.

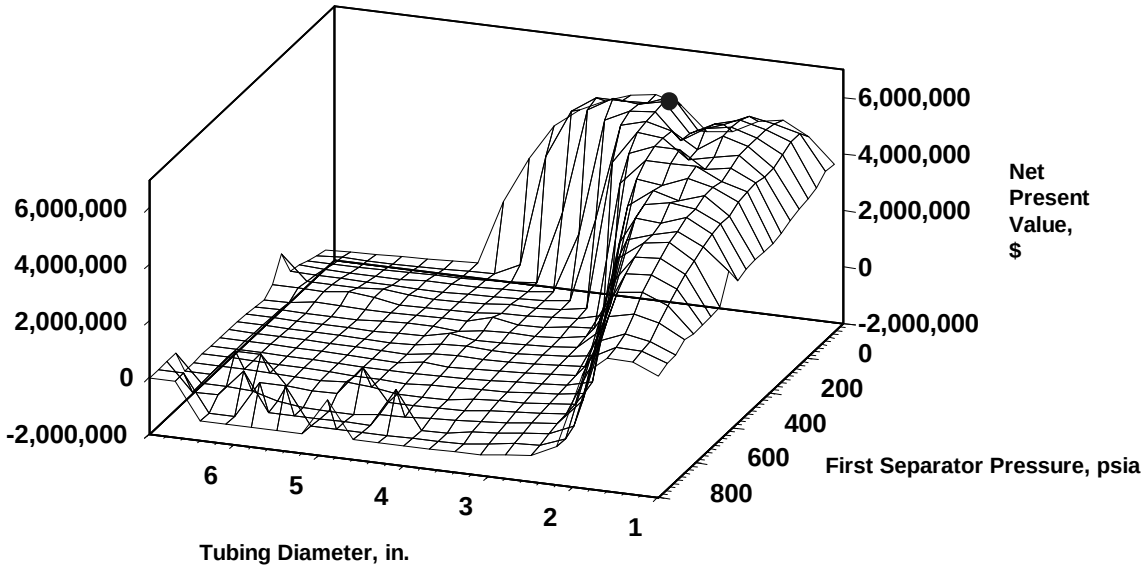

Figure 4.1: Surface of Problem 1

4.1.1 Newton Type Solution

The Newton type technique used in this project was the Marquardt modification of Newton's method, with a line search. The rough nature of this surface prevented this approach from being particularly successful. Table 4.2 describes each of five runs of this technique. Notice that the number of function calls was large due to the expensive Hessian matrix construction process using finite differences.

Table 4.2: Marquardt Solution, Problem 1

Modified Marquardt Solution						
Start	(2.0,500)	(3.5,325)	(4.5,200)	(5.5,700)	(6.5,300)	Average
Final Solution	(1.48,160.3)	(2.69,157.2)	(4.5,200)	(6.86,700.0)	(6.89,250.0)	N/A
Value	5,554,826	6,233,728	-905,823	-905,309	-882,710	1,818,942
Function Calls	41	154	6	20	27	49.6

Note that the success rate was not high, as four of the five attempts failed. Only the second run converged to a near-optimal solution. The paths followed during each of these five attempts is illustrated in Figure 4.2.

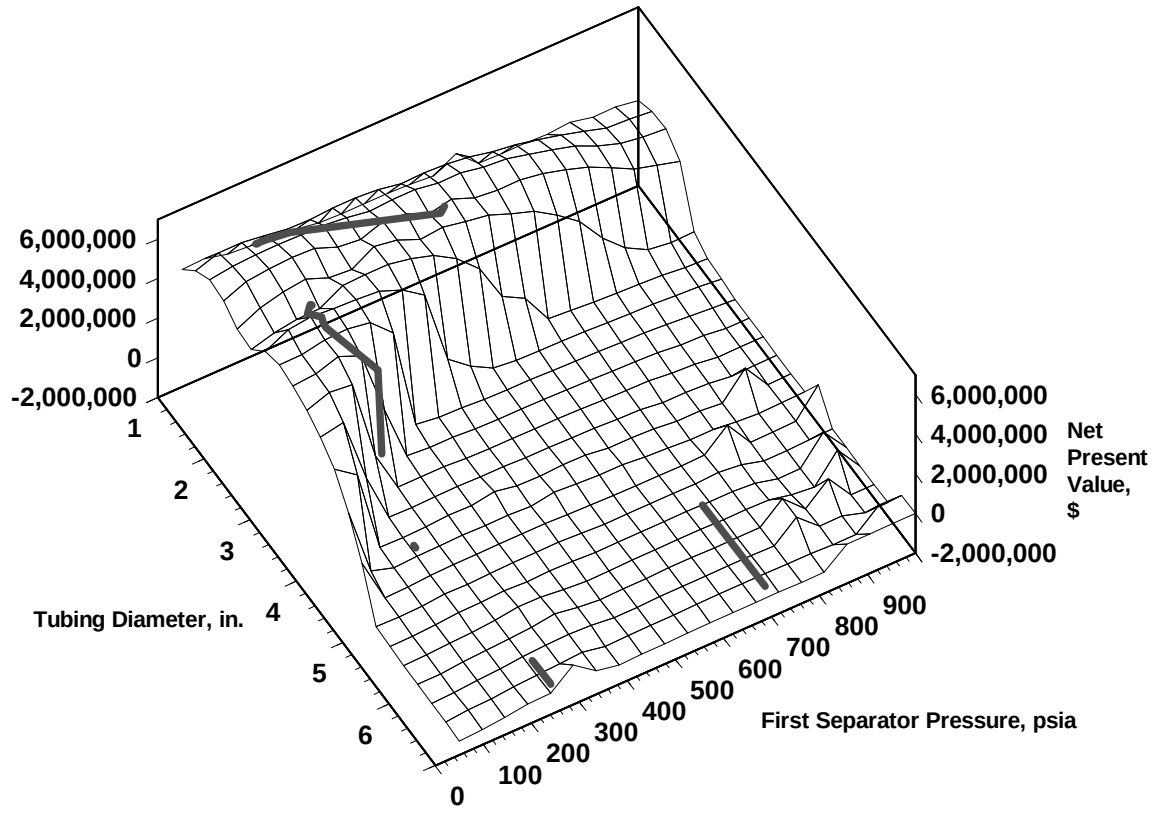

Figure 4.2: Newton Method Paths

4.1.2 Polytope Algorithm Solution

The polytope algorithm discussed in Section 3 was applied to the same five starting points as the Newton type method. The polytope did not succeed for all starting points, but it was much more successful than the Newton type method. Three of the starting points climbed to similar optima, while two points became stuck in local maxima. The paths of the best point for each polytope are shown in Figure 4.3.

Table 4.3: Polytope Solutions, Problem 1

Polytope Solution						
Start	(2.0,500)	(3.5,325)	(4.5,200)	(5.5,700)	(6.5,300)	Average
Final Solution	(3.05,174.0)	(2.99,192.2)	(2.69,164.1)	(7,700)	(7,357.4)	N/A
Value	6,244,684	6,240,147	6,241,428	-907,485	-752,226	3,413,309
Function Calls	64	55	34	14	26	38.6

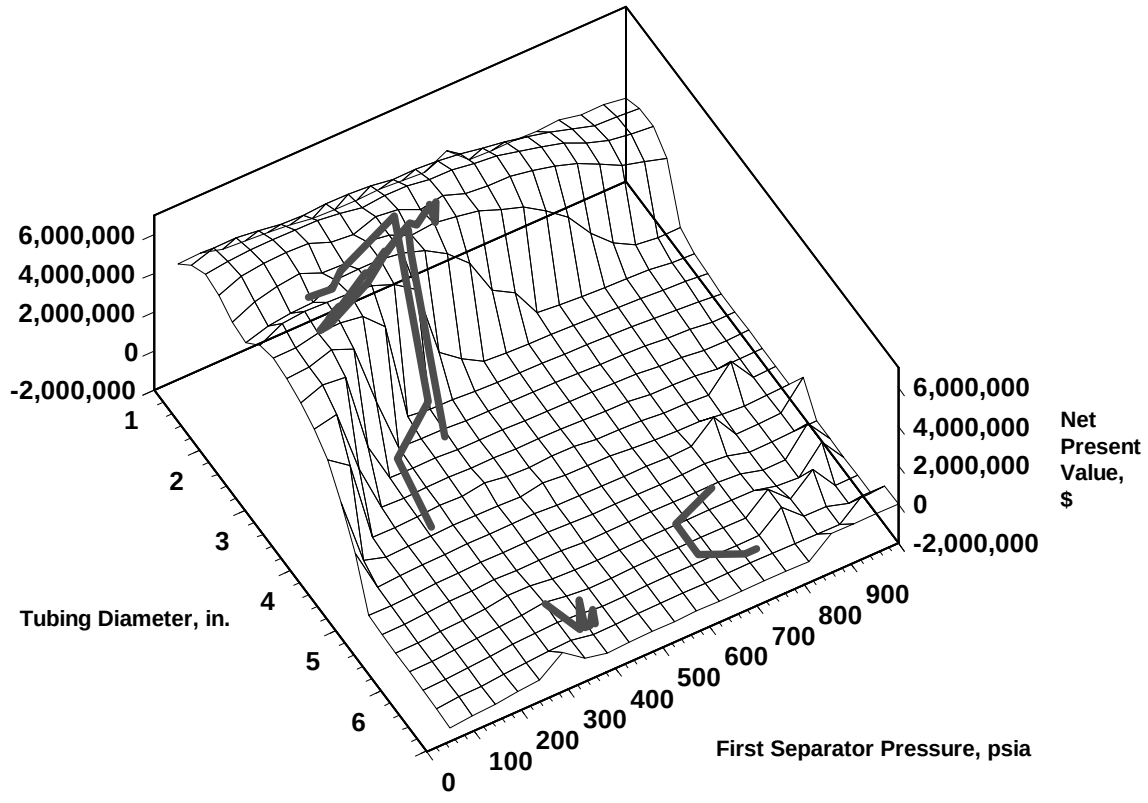

Figure 4.3: Polytope Solution Paths

4.1.3 Genetic Algorithm Solution

This problem was used to test a variety of genetic algorithm approaches. Each of the two variables was treated as a discrete variable with 64 evenly spaced possible values. Hence, there were two 6 bit variables, for a total string length of 12 bits. This search space included a total of 4,096 possible combinations. Each generation had a population of 12 members. For each case, the algorithm was tested with 5 different random seeds. The benchmarks used for comparison are the average final objective function value, and the number of function calls needed to converge. In the course of making these runs, the search space was sampled completely, and the maximum was found to occur at a tubing diameter of 3.10 inches and a first separator pressure of 155.56 psia. This combination of parameters produced a net present value of $6,233,859.

4.1.3.1 Mutation Rates

The first genetic algorithm parameter studied was the mutation rate. A variety of mutation rates were used. Up to two mutations were allowed per member each generation. The first term in the mutation description is the probability that only one mutation will occur. The second term is the probability that two mutations will occur. The efficiency of different mutation rates is outlined in Table 4.4.

Table 4.4: Mutation Rate Effects

Mutation Rate, *no other modification*				
Mutation Rate (1st, 2nd)	(0.05,0.0)	(0.5,0.0)	(1.0,0.0)	(0.5,0.5)
Average 100th Gen Solution	6,029,982	6,228,461	6,227,651	6,230,058
Function Calls to Achieve Solution	38.6	189.0	359.4	320.2

In this case, the lowest mutation rate did not achieve the desired convergence within 100 generations. The rates higher than fifty percent chance of one mutation performed well, but they required a lot of function calls. Thus, the base rate used for the rest of the cases was fifty percent chance for one mutation, no chance for second mutation.

4.1.3.2 Best Member Extrapolation

Another modification of genetic algorithms is best member extrapolation. This hybrid technique involves using the best two different members of the population to establish another population member. The idea is that this will allow the genetic algorithm to capitalize on trends in the objective function surface. The idea is illustrated in Figure 4.4.

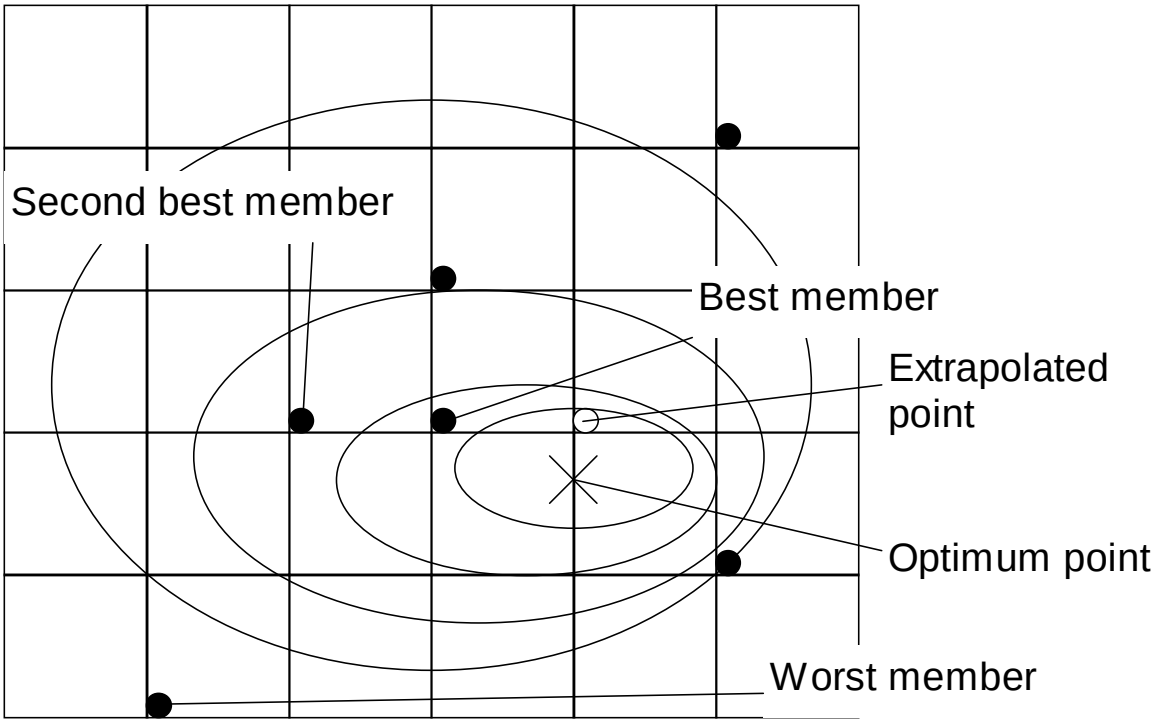

Figure 4.4: Illustration of Best Member Extrapolation

Table 4.5: Best Member Extrapolation Results, *(0.5,0) mutation rate*

Best member extrapolation, *(0.5,0.0) mutation rate*		
Best Member Extrapolation	No	Yes
Average 100th Gen Solution	6,228,461	6,206,010
Function Calls to Achieve Solution	189.0	176.0

Extrapolation was not useful for this mutation rate. The modification was also tested with higher mutation rates. It is conceivable that a different mutation rate will provide more opportunities for extrapolation, as explored in Table 4.6.

Table 4.6: Best Member Extrapolation Results, *(0.5,0.5) mutation rate*

Best member extrapolation, (0.5,0.5) mutation rate		
Best Member Extrapolation	No	Yes
Average 100th Gen Solution	6,230,058	6,232,352
Function Calls to Achieve Solution	320.2	436.4

In this case, extrapolation provided better final results, but at the cost of more function calls. Extrapolation may prove useful in other cases as well.

4.1.3.3 Bit Crossover vs. Variable Crossover

Variable crossover was another genetic algorithm modification tested. The standard genetic algorithm allows crossover to occur at a random bit in the string. As the crossover may occur at any point of a variable, it may amount to a type of mutation. It is reasonable to believe that crossovers occurring after a randomly drawn variable, rather than a randomly drawn bit, might preserve good variable values. The following table compares the results for two types of crossover.

Table 4.7: Bit Crossover vs. Variable Crossover Results, *(0.5,0.0) mutation rate*

Crossover Type, (0.5,0.0) mutation rate		
Crossover Type	Random Bit	Random Variable
Average 100th Gen Solution	6,228,461	6,143,373
Function Calls to Solution	189.0	159.6

Because this problem has only two variables, the crossover always occurred at the end of the first variable. In this case, variable rather than bit crossover caused distinctly worse results. Furthermore, in two of the five runs, premature convergence occurred. Use of random variable crossover would probably have to be compensated for by increasing the rate of mutation.

4.1.3.4 Mutation Type

In addition to the rate of mutation, there is also the question of mutation type. The standard genetic algorithm technique of switching the value of a random bit is a good

method of exploring the search space, but it may waste function calls when the algorithm has already found the best region in which to concentrate. It is possible to change the way mutation occurs after a number of generations. Instead of changing a random bit, the binary value of a randomly selected variable will be increased or decreased by one. This allows for a more careful search around the already determined best region in the search space. Table 4.8 compares the results for different switch-over generation. Before the switch-over generation, mutation occurs in the classic fashion. After the switch-over generation, mutation occurs in the +/- 1 fashion.

Table 4.8: Mutation Switch-Over Results, *(0.5.0.0) mutation rate*

Mutation Change, (0.5,0.0) mutation rate					
Change Step	never	75	50	25	0
Average 100th Gen Solution	6,228,461	6,230,058	6,230,058	6,232,352	6,108,170
Function Calls to Solution	189.0	227.8	209.4	142.8	107.8

Clearly, this represents an improvement over the classic mutation. A switch-over at 25 generations provides sufficient function calls for exploration of the search space, and leads to good results with a low number of function calls. The next idea studied was combining this type of mutation with best member extrapolation.

Table 4.9: Mutation Switch-Over With Best Member Extrapolation, *(0.5,0.0) mutation rate*

Mutation Change with extrapolation, (0.5,0.0) mutation rate					
Extrapolation	No			Yes	
Change Step	never	50	25	50	25
Average 100th Gen Solution	6,228,461	6,230,058	6,232,352	6,227,831	6,229,338
Function Calls to Solution	189.0	209.4	142.8	199.4	170.8

This modification did not work with extrapolation for this mutation rate. However, when the mutation rate was increased, the results are presented in Table 4.10.

Table 4.10: Mutation Switch-Over With Best Member Extrapolation, *(0.5,0.5) mutation rate*

Mutation Change with extrapolation (0.5,0.5) mutation rate			
Change Step	never	25 no ex	25 with ex
Average 100th Gen Solution	6,230,058	6,233,859	6,233,859
Function Calls to Solution	320.2	241	218.4

While the final results were perfect both with and without extrapolation, extrapolation seemed to lower the number of function calls.

4.1.3.5 Culling

Another modification that focuses on improvement after an initial search of the space has been completed is culling. After a fixed number of generations have passed, the fitness table is modified to eliminate a fixed number of the members. In this project, the effects of culling 50 percent of the population was studied.

Table 4.11: Culling Effects, *(0.5,0.0) mutation rate*

Culling 50 Percent Effects, (0.5,0.0) mutation rate					
Begin Culling Step	never	75	50	25	0
Average 100th Gen Solution	6,228,461	6,228,461	6,228,461	6,216,890	6,074,318
Function Calls to Solution	189.0	189.0	182.4	132.6	65.2

The culling results are ambiguous, but culling does not seem to be particularly promising. For this project, culling was not considered further.

4.1.3.6 Fitness Scaling

Scaling the fitness of the members is a method of giving additional weight to the weakest members. This may prove beneficial in cases exhibiting premature convergence behavior, as it prevents the fittest members in early generations from monopolizing the breeding opportunities. There are several ways to use scaling. For this project, without scaling, the net present value of the worst member was taken away from the net present value of each member. These values were totaled, and the

fitness of each member was its fraction of the total value. This was done, because it was possible to have negative objective function values. Of course, it completely eliminated the worst members genetic material by giving it no breeding opportunities. In order to enact scaling, after the worst member's value was subtracted, a constant value was added to each member. In this scenario, even the worst member has a chance to breed. The constant value added was 25 percent of the total unscaled fitness. Table 4.12 illustrates the results of scaling.

Table 4.12: Fitness Scaling Results, *(0.5,0.0) mutation rate*

Fitness Scaling Results, (0.5,0.0) mutation rate		
Scaling	No	Yes
Average 100th Gen Solution	6,228,461	6,226,978
Function Calls to Solution	189.0	155

No clear benefit of scaling was demonstrated. Scaling was not considered further.

4.1.3.7 Genetic Algorithm Parameters Summary

Studying all of these parameters and modifications resulted in the conclusion that, for this problem, a good set of genetic algorithm parameters includes a high mutation rate, mutation switching, and best member extrapolation. The paths of these solutions is illustrated in Figure 4.5 (the paths shown are the best point from each generation). The set of parameters listed in Table 4.13 produced these paths, and was used for all other genetic algorithm runs in this project.

Table 4.13: Efficient Set of Genetic Algorithm Parameters

Parameter	**Values/Utilize?**
mutation rate - one mutation	50%
mutation rate - two mutations	50%
crossover type	random bit
best member extrapolation	yes
mutation change generation	25
culling	no
fitness scaling	no

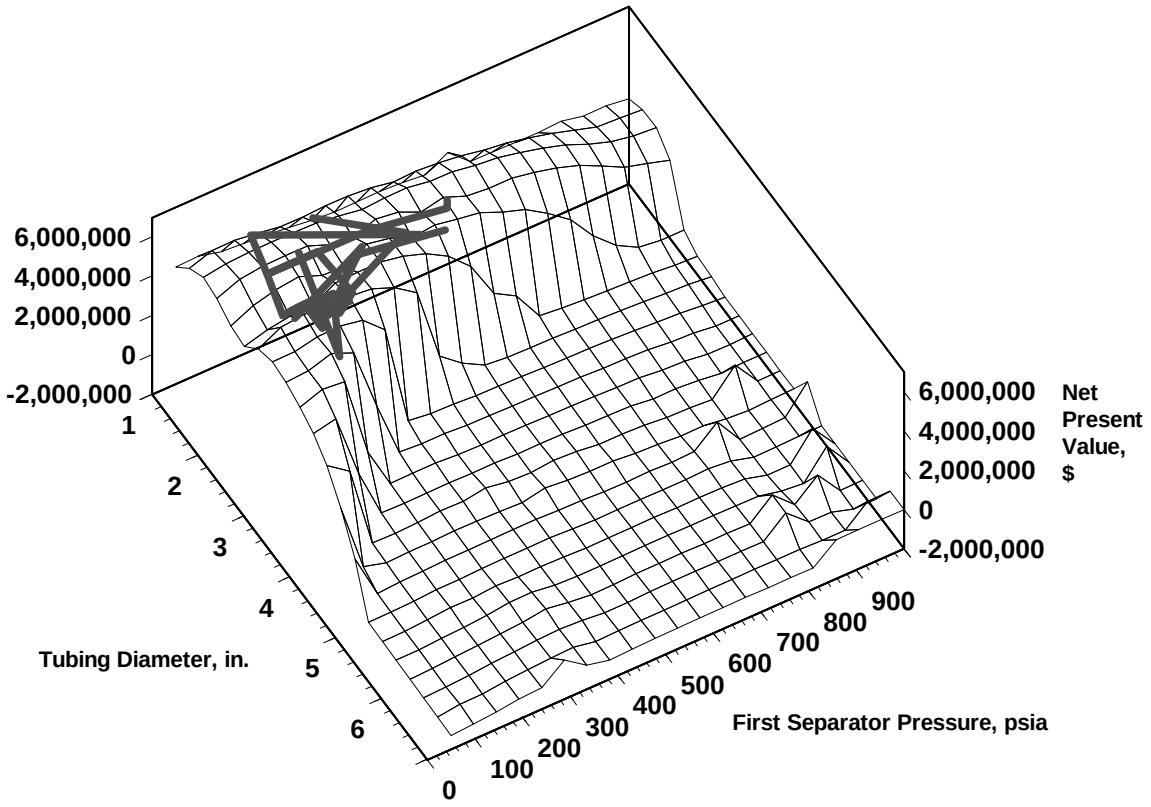

Figure 4.5: Genetic Algorithm Paths, Problem 1

4.1.4 Problem 1, Comparison of Methods

For this problem, both the polytope and the genetic algorithm worked well. The Newton type method was not successful, largely due to the rough nature of the surface. The Newton type method was not used to solve other problems. The polytopes proved to be much quicker than the genetic algorithm, but not nearly as reliable. If a polytope starts at a good position, it will probably quickly converge to the proper solution. On the other hand, an engineer may not always know a good starting point for optimization algorithms. The other alternative is to start several polytopes from different positions. This approach would be more certain, but would probably be no cheaper than the genetic algorithm. Figure 4.6 shows the convergence behavior of the polytope and the genetic algorithm runs. Note that the two runs which failed were both polytopes which unfortunately began too far from the correct solution.

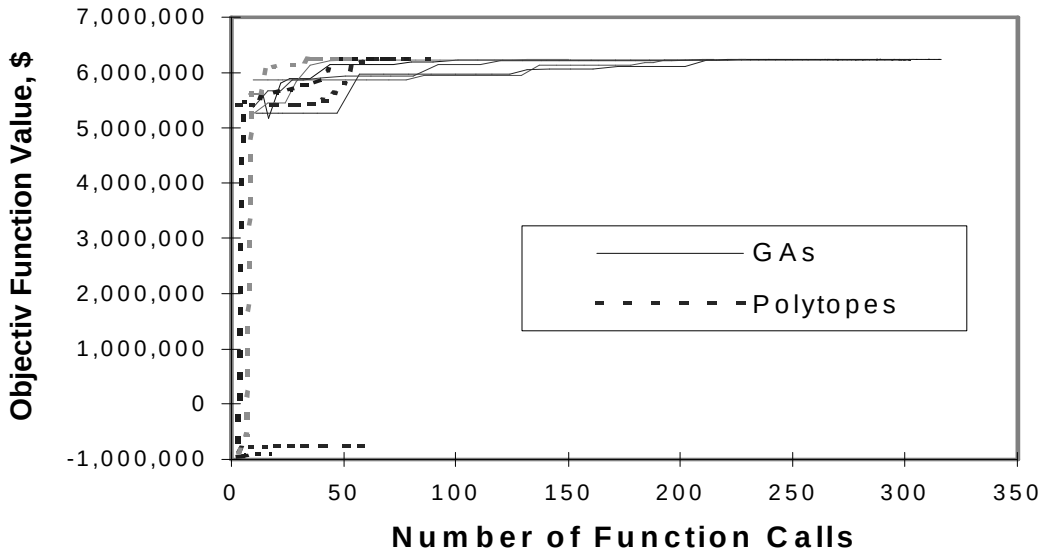

Figure 4.6: Polytope and Genetic Algorithm Performance

4.2 Problem 2

This problem is a modification of Problem 1. The composition was altered to represent a lighter, more volatile fluid, and the optimization procedures were repeated. Specifically, the best combination of genetic algorithm parameters and the five polytope runs were repeated. The composition is described in Table 4.14. The variables are first separator pressure and tubing diameter. There is no time dependence for these variables. This is the base run used in Section 5 for the sensitivity study.

Table 4.14: Reservoir Fluid, Problem 2

Reservoir Fluid						
C_1	C_2	C_3	C_5	C_8	C_{10}	C_{11+}
0.09	0.13	0.12	0.13	0.19	0.19	0.15

The surface of the objective function evaluated for this problem is illustrated in Figure 4.7. The surface is similar to the original surface, but is significantly different. The optimum, as determined by the polytope routine, now occurs at a tubing diameter of 2.98 inches, and a first separator pressure of 207.8 psia. This combination of parameters corresponds to a NPV of $7,114,896. The surface is much rougher in

appearance than the surface of Problem 1. Additionally, the plateau in the region of the optimum is larger.

The genetic algorithm was run with five different random seeds, producing a family of curves to compare. Thus, for the polytope technique, five random locations were used. Because the Newton technique had failed for Problem 1, it was not used on Problem 2.

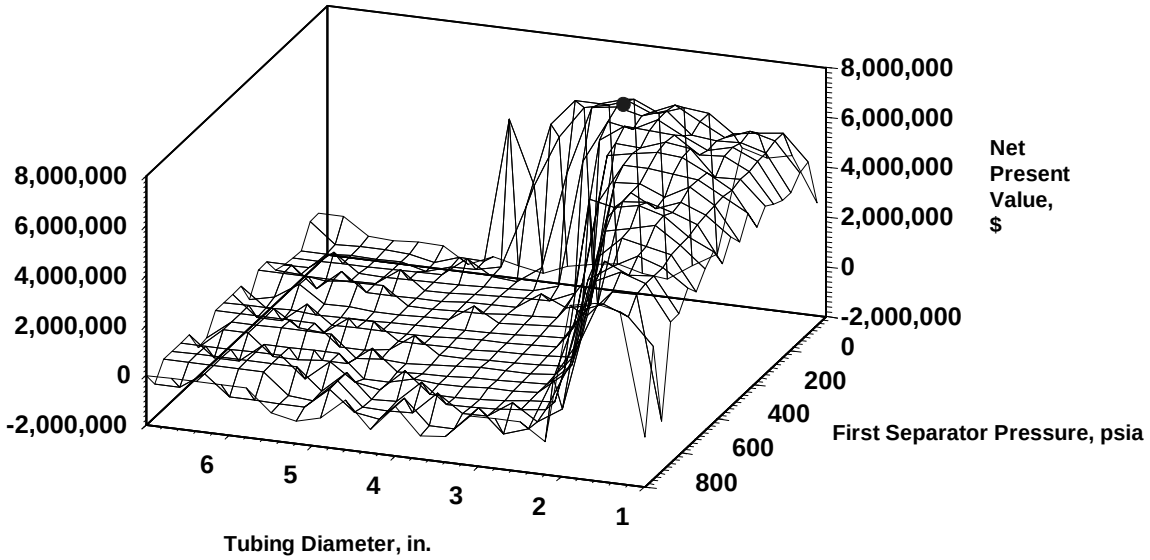

Figure 4.7: Objective Function Surface, Problem 2

4.2.1 Polytope Algorithm Solution

The polytope algorithm was applied to the same five starting points as in Problem 1. The polytope did not succeed for each starting point. Three of the starting points climbed to similar optima, while two points became stuck in local maxima. Each of the polytope runs is described in Table 4.15.

Table 4.15: Polytope Solutions, Problem 2

Polytope Runs						
Start	(2.0,500)	(3.5,325)	(4.5,200)	(5.5,700)	(6.5,300)	Average
Final Solution	(2.98,207.8)	(2.5,193.75)	(3,212.5)	(7,643.75)	(7,375)	N/A
Value	7,114,896	6,972,582	7,068,818	-749,835	-512,980	3,978,696
Function Calls	68	36	48	28	28	41.6

4.2.2 Genetic Algorithm Solution

The genetic algorithm was used with the parameters described in Table 4.13. Each run converged to the same solution, which was better than any solution found by the polytope routine.

Table 4.16: Genetic Algorithm Solutions, Problem 2

Genetic Algorithm Solutions, Parameters Described in Table 4.13						
Random Seed	14,307	18,232	6,125	5,515	15,948	avg.
100th Gen Solution	(3,200.79)	(3,200.79)	(3,200.79)	(3,200.79)	(3,200.79)	(3,200.79)
Value	7,122,886	7,122,886	7,122,886	7,122,886	7,122,886	7,122,886
Function Calls	272	221	220	310	211	246.8

4.2.3 Problem 2, Comparison of Methods

The genetic algorithms were again shown to be stable and efficient. While polytope solutions converged rapidly, they did not always converge to the correct solution. Again, the choice of genetic algorithm parameters seems to be very effective.

4.3 Problem 3

Problem 1 and Problem 2 had only two decision variables each. However, many interesting petroleum engineering optimization problems deal with more than two decision variables. Problem 3 features the same reservoir as the first two problems, but has eight decision variables. In this case, the engineer is controlling tubing diameter and gas injection rate. There are four time periods, so the model must deal with a total of eight decision variables. All other features of the problem are enumerated in Table 4.17.

Table 4.17: Problem 3 Description

Reservoir						
Areal Extent of Reservoir (acres)					160.	
Thickness (feet)					50.	
Porosity					0.30	
Initial Res. Pressure					3,500.00	
Reservoir Temperature (^{o}Rankine)					650.	
Permeability (md)					5.0	
S_{wc}, S_{or}, S_{gr}					0.0, 0.3, 0.0	
n_o, n_g					1.5,1.0	
Reservoir Depth					8,000	
Reservoir Fluid						
C_1	C_2	C_3	C_5	C_8	C_{10}	C_{11+}
0.05	0.13	0.12	0.13	0.19	0.19	0.19
Production Parameters (non-variable)						
first separator pressure (psia)					150.0	
choke diameter (1/64 inches),					30.0	
surface line diameter (inches),					1.0	
P_{sep2}/P_{sep1}					0.95	
P_{sep3}/P_{sep2}					0.90	

As in Problem 2, the Newton type technique was not used to solve this problem.

4.3.1 Polytope Algorithm Solution

The polytope was run twice with no modification. The first run was from a completely random location. The second run was from a location exactly in the middle of the search space. The results are listed in Table 4.18.

Table 4.18: Polytope Solutions, Problem 3

Sol'n		Tubing Diameter inches				Injection Rate Mscf/D				Function Calls	NPV $
	years	0-5	5-10	10-15	15-20	0-5	5-10	10-15	15-20		
1	*start*	4.5	2.75	6	3.5	1300	800	1800	100	0	1,368,043
	end	1.87	5.77	6.08	5.72	703.5	805.9	1368	41.2	167	4,228,211
2	*years*	0-5	5-10	10-15	15-20	0-5	5-10	10-15	15-20		
	start	4	4	4	4	1000	1000	1000	1000	0	3,452,439
	end	3.73	5.02	5.33	5.38	1042	579.8	621.6	644.7	369	4,859,343

Notice that the two solutions vary considerably. By comparison to the genetic algorithm solution (presented next), these solutions are not nearly optimal. Part of the problem must be in selection of initial starting point. We may be able to relieve part of this problem by temporarily simplifying Problem 3. The problem was run without the time variability of the decision variables. The results of that two variable optimization were then used as the start of a four variable problem (two time periods). Finally, the results of the four variable optimization were used as the start of an eight variable problem (four time periods). The results of this optimization are shown in Table 4.19.

Table 4.19: Multistep Polytope Solution, Problem 3

# Dec. Vars.		Tubing Diameter inches				Injection Rate Mscf/D				Cum. Function Calls	NPV $
	years	0-5	5-10	10-15	15-20	0-5	5-10	10-15	15-20		
2	*start*	4	4	4	4	1000	1000	1000	1000	0	3,452,439
	end	1.984	1.984	1.984	1.984	133.6	133.6	133.6	133.6	76	6,381,308
4	*years*	0-5	5-10	10-15	15-20	0-5	5-10	10-15	15-20		
	start	1.984	1.984	1.984	1.984	133.6	133.6	133.6	133.6	76	6,381,308
	end	1.984	1.984	2.734	2.734	133.6	133.6	133.6	133.6	187	6,528,051
8	*years*	0-5	5-10	10-15	15-20	0-5	5-10	10-15	15-20		
	start	1.984	1.984	2.734	2.734	133.6	133.6	133.6	133.6	187	6,528,051
	end	1.984	1.984	2.734	2.734	133.6	133.6	133.6	133.6	302	6,528,051

Clearly this approach worked much better than simply starting the polytope with the full regimen of decision variables. However, this sort of strategy may not always be possible. If, for example, there are eight decision variables with no time dependence, there is no analogous strategy. Notice that the last step did not produce any improvement. This indicates that polytopes may be more appropriate for problems with a small number of decision variables.

4.3.2 Genetic Algorithm Solution

This problem was solved with the genetic algorithm one time. The solution is described in Table 4.20. To reach this solution, 3,120 function calls were required. Each of the eight variables was optimized with a six bit string. This means that the search space includes 2^{48} ($\approx 10^{14.45}$) possible members. The number of calls required, 3,120, is not a significant percentage of the possible function calls. On the other hand, there is no proof that this solution is the global optimum. Regardless, this solution is better than any of the polytope solutions.

Table 4.20: Genetic Algorithm Solution, Problem 3

Tubing Diameter inches				Injection Rate Mscf/D				Cum. Function Calls	NPV
0-5	5-10	10-15	15-20	0-5	5-10	10-15	15-20		
1.76	3.38	2.90	2.62	126.9	222.2	158.7	126.9	3,120	6,688,242

4.3.3 Problem 3, Comparison Of Methods

Problem 3 showed that the polytope method is not necessarily convergent for this sort of problem. The best polytope solution, the multistep approach, achieved an objective function value that is 97.6 percent of the solution provided by the genetic algorithm. The polytope achieved this solution with 187 function calls, compared to the 3,120 required by the genetic algorithm. Still, this sort of multistep approach might not be possible for every problem. The genetic algorithm proved to be robust and stable.

5. Reservoir Parameter Sensitivity Study

The optimization techniques described in Section 4 are useful for finding the best combination of engineering parameters for a model with a given set of reservoir parameters. However, it is unusual for a petroleum engineer to know the value of a reservoir parameter, such as porosity or permeability, to a high degree of certainty. It is more common for the engineer to know a distribution, or range of parameter values. In general, certainty in a parameter can only be purchased at the price of expensive well testing, logging, or coring. Engineers face difficulty in determining which tests to spend resources. This project established that optimization techniques provide an opportunity for quantifying the value of greater certainty regarding reservoir data.

If an engineer selects a set of engineering parameters to optimize based on a given set of reservoir parameters, it is logical to ask what will happen if a reservoir parameter does not have the expected value. Typically, changes in reservoir parameters will have intuitively predictable changes in the objective function value (NPV). For instance, if permeability is greater than expected, the NPV will be greater. What is not as intuitive is how optimized engineering parameters vary with the reservoir parameter. For example, how does the optimized first separator pressure change if the permeability is greater than expected. It is likely that the engineering parameters optimized for the expected value will be suboptimal for the realized reservoir parameter value.

If the reservoir parameter has a value other than the expected value, it is tempting to describe the loss (or gain) as the difference between the optimized NPV for the realized value and the optimized NPV for the expected parameter value. For the purpose of determining the value of reservoir certainty, this is not correct. In fact, the loss should be the difference between the optimized NPV for the realized reservoir parameter value, and the NPV based on the realized parameter value, produced with the optimized engineering parameters for the expected reservoir parameter value. Figure 5.1 illustrates this principle through a hypothetical case. The expected reservoir parameter (permeability) value is 10 md. The engineer plans for this permeability, and predicts an NPV of A. Instead, the reservoir parameter value is 1 md, and even with perfect knowledge, the field NPV could only be B. Instead, however, the field is

produced with a suboptimal separator pressure, one which would be optimal for 10 md. This results in a NPV of C. In this case, the cost of uncertainty in the reservoir parameter is the difference between the optimal and suboptimal NPVs for 1 md.

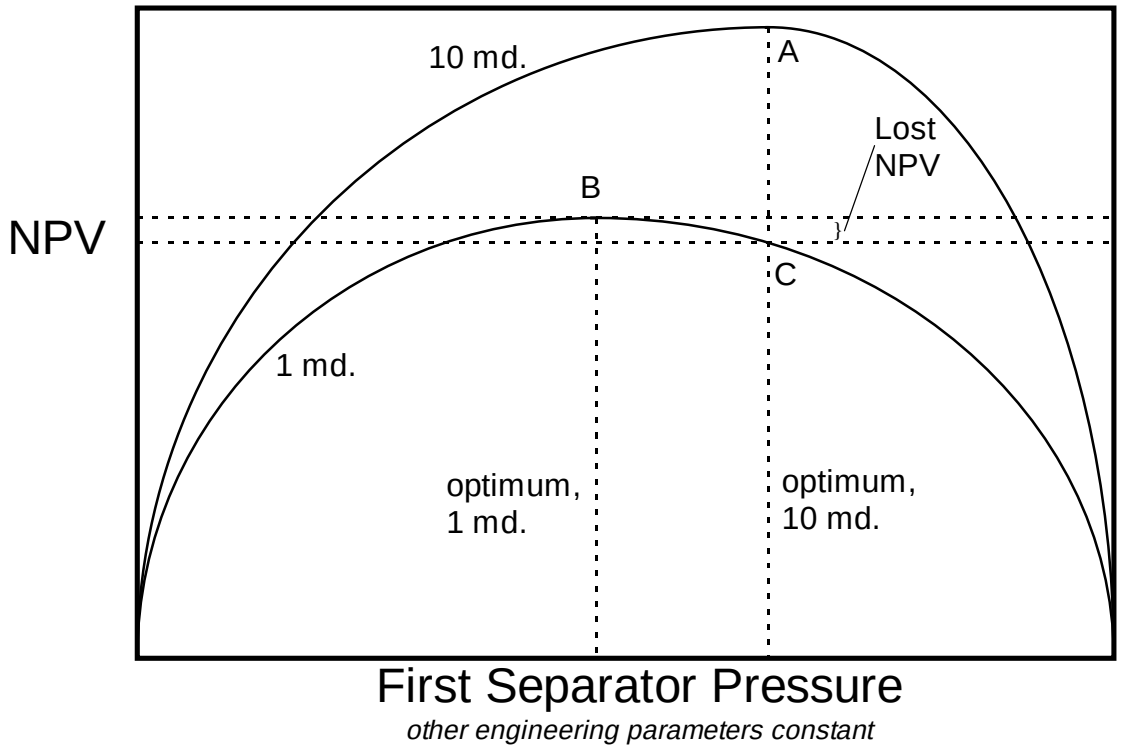

Figure 5.1: Lost NPV Due to Uncertainty

Using Problem 2 from Section 4 as a base case, this principle is used to study the sensitivity of optimized parameters to reservoir uncertainty. Four reservoir parameters were analyzed. The cost of uncertainty is established by combining the probability distribution with the discrepancy between optimized NPV and the realized suboptimal NPVs. For the sensitivity study of each parameter, all other reservoir parameters were held constant. Thus, each sensitivity study is performed in a univariate context.

5.1 Composition

The composition was varied by altering the mole percent of methane (C_1). As the methane quantity varied, the mole percent of C_{11+} was changed to balance. There is a twenty percent chance that the methane percentage falls between 0.0 and 3.0. There is a twenty percent chance that the value is between 3.0 and 7.0. The third quintile

falls between 7.0 and 11.0 percent. The fourth quintile contains the range of 11.0 to 15.0 percent. There is also a twenty percent chance that the methane value falls from 15.0 to 100.00 percent. Table 5.1 describes this distribution.

Table 5.1: Distribution of Mole Percent Methane

Cumulative Distribution Function, Minimum	Cumulative Distribution Function, Minimum	Minimum Mole Percent Methane	Maximum Mole Percent Methane
0.0	0.2	0.0	3.0
0.2	0.4	3.0	7.0
0.4	0.6	7.0	11.
0.6	0.8	11.	15.
0.8	1.0	15.	100.

For each quintile, a representative mole percentage of methane was chosen, and the model was optimized for each representative mole percentage. These representative values are enumerated in Table 5.2. The optimization was performed with the genetic algorithm parameters used for Problem 2 (see Table 4.13).

Table 5.2: Representative Mole Percent Methane Values for Optimization

Cumulative Distribution Function, Minimum	Cumulative Distribution Function, Minimum	Representative Mole Percent Methane	Probability
0.0	0.2	1.0	20%
0.2	0.4	5.0	20%
0.4	0.6	9.0	20%
0.6	0.8	13.	20%
0.8	1.0	17.	20%

Table 5.3 contains the results of these optimization runs. Notice that the optimized engineering parameters changed as the mole percent methane changed. Figure 5.2 illustrates the path taken by the optimal parameters as the mole fraction changed.

Table 5.3: Optimization Results - Composition Distributed

Representative Mole Percent Methane	Optimum Values		Optimal NPV
	Tubing Diameter, inches	Separator Pressure, psia	
1.0	3.19	125.4	5,269,483
5.0	3.1	155.6	6,233,859
9.0	3.0	200.8	7,122,886
13.0	2.9	215.9	7,991,055
17.0	2.71	215.9	8,694,420

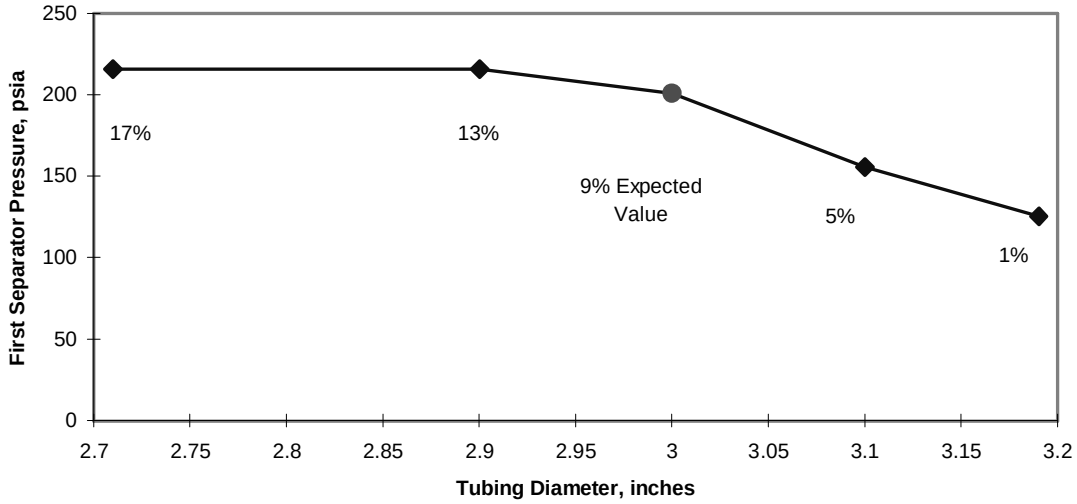

Figure 5.2: Optimal Parameter Path - Composition

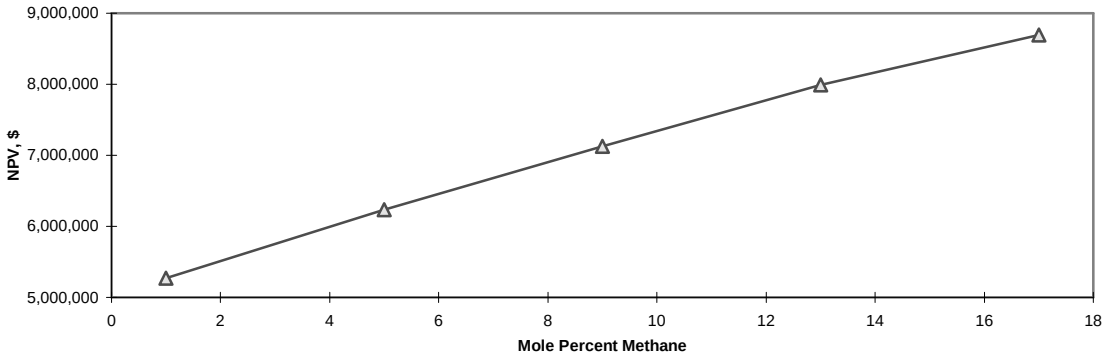

Figure 5.3: NPV vs. Composition

There is a clear trend in both the NPV and the optimized parameters. However, Figure 5.3 represents NPV given perfect information. The interesting information is the NPV for each reservoir composition if produced at the optimized engineering parameters for the expected mole fraction of methane. Table 5.4 lists the NPV produced using the suboptimal engineering parameters.

Table 5.4: NPV if Engineering Parameters Optimized for Expected Composition

Representative Mole Percent Methane	NPV tubing - 3.0"; sep pressure - 200.8 psia
1.0	4,910,630
5.0	6,184,787
9.0	7,122,886
13.0	7,956,254
17.0	8,347,179

The real value of additional certainty in the composition is the difference in the NPVs listed in Table 5.3 and Table 5.4. These differences are illustrated in Figure 5.4. Not surprisingly, the further the composition varies from the expected value, the greater the discrepancy between the NPVs. These differences must be weighted by the probabilities listed in Table 5.2. This operation is demonstrated in Table 5.5. In this case, the cost of uncertainty in composition is $157,993. Any choice to spend money on testing composition should be measured against this cost.

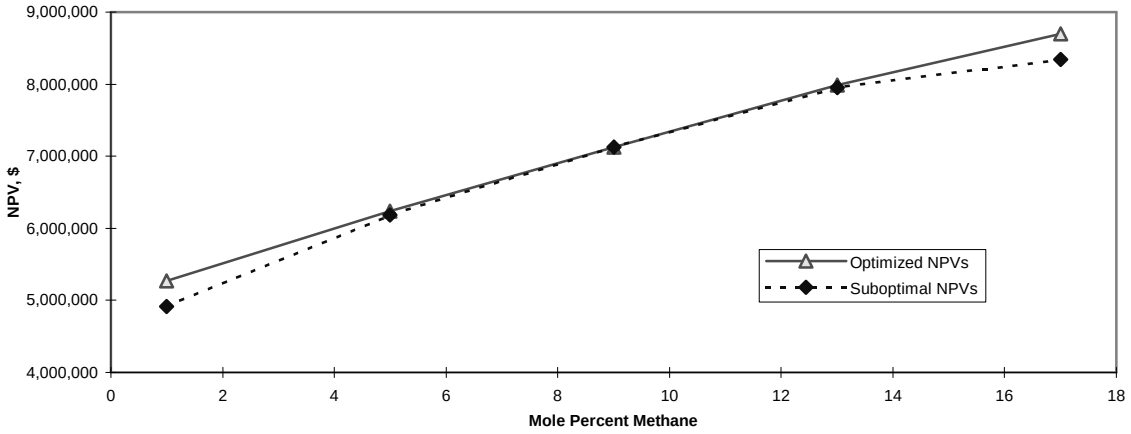

Figure 5.4: Discrepancy in NPV vs. Composition

Table 5.5: Value of Certainty in Composition

Representative Mole Percent Methane	Optimal NPV	NPV tubing - 3.0"; sep pressure - 200.8 psia	Discrepancy	Probability	Cost of Uncertainty
1.0	5,269,483	4,910,630	358,853	20%	71,771
5.0	6,233,859	6,184,787	49,072	20%	9,814
9.0	7,122,886	7,122,886	0	20%	0
13.0	7,991,055	7,956,254	34,801	20%	6,960
17.0	8,694,420	8,347,179	347,241	20%	69,448
				Sum	157,993

5.2 Permeability

The permeability is distributed as described in Table 5.6. Five percent of permeabilities are less than 1.256 md. One quarter percent of the permeabilities fall between 1.256 md and 3.155 md. Forty percent of the permeabilities lay between 3.155 md and 7.924 md. Another 225 percent of the permeabilities are greater than 7.924 and less than 19.905 md. The final five percent of the permeabilities are greater than 19.905 md.

Table 5.6: Distribution of Permeabilities

Cumulative Distribution Function, Minimum	Cumulative Distribution Function, Minimum	Minimum Permeability md	Maximum Permeability md
0.0	0.05	0.0	1.256
0.05	0.3	1.256	3.155
0.3	0.7	3.155	7.924
0.7	0.95	7.924	19.905
0.95	1.0	19.905	100.

For each of these ranges, a representative value was selected, and the engineering parameters were optimized for those representative values. The representative permeabilities are presented in Table 5.7. Table 5.8 presents the results of the optimization runs. Figure 5.5 illustrates the path the optimum engineering parameters

take as the permeability is changed. In this case, the path is more scattered than it was for changes in composition. The changes in NPV vs. permeability are illustrated in Figure 5.6.

Table 5.7: Representative Permeabilities for Optimization

Cumulative Distribution Function, Minimum	Cumulative Distribution Function, Minimum	Representative Permeability, md	Probability
0.0	0.05	0.792	5%
0.05	0.3	1.99	25%
0.3	0.7	5.0	40%
0.7	0.95	12.56	25%
0.95	1.0	31.55	5%

Table 5.8: Optimization Results - Permeability Distributed

Representative Permeability md	Optimum Values		Optimal NPV
	Tubing Diameter, inches	Separator Pressure, psia	
0.792	2.9	336.51	2,957,346
1.99	3.1	246.03	5,177,912
5.0	3.0	200.79	7,122,886
12.56	3.57	155.56	9,657,152
31.55	3.29	125.4	12,394,272

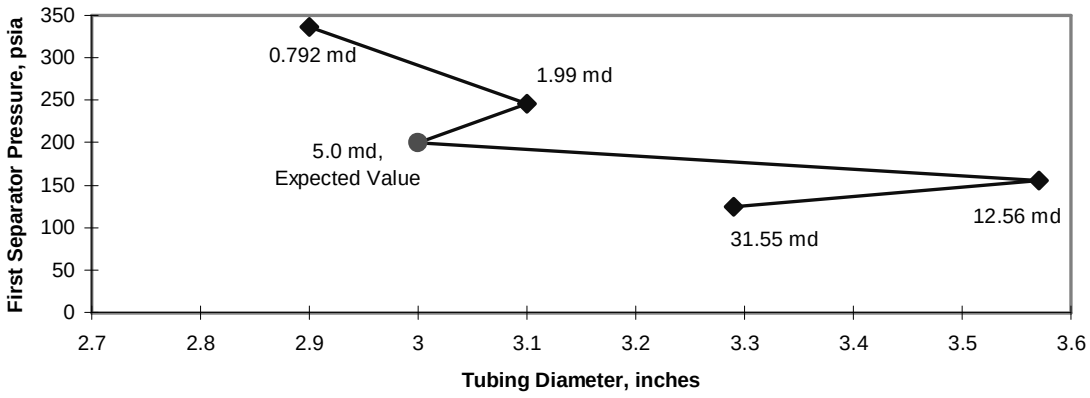

Figure 5.5: Optimal Parameter Path - Permeability

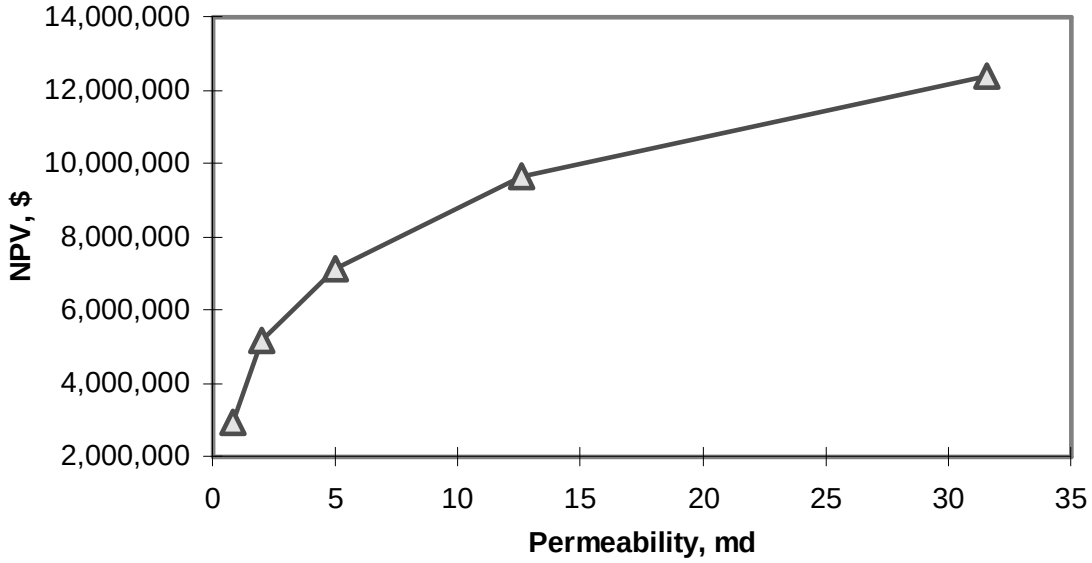

Figure 5.6: NPV vs. Permeability

Again, the interesting information is how these optimized NPVs compare to the NPVs based on the engineering parameters optimized for the expected permeability. These NPVs are listed in Table 5.9. Figure 5.7 demonstrates the discrepancy between the optimal and suboptimal NPVs. The cost of uncertainty was calculated to be $388,203, as is illustrated by Table 5.10. Costs of narrowing the uncertainty in permeability can be measure against this value. Furthermore, the relative merits of permeability vs. composition testing can be found by comparing this number to the value calculated for composition.

Table 5.9: NPV if Engineering Parameters Optimized for Expected Permeability

Representative permeability	NPV tubing - 3.0"; sep pressure - 200.8 psia
0.792	1,874,792
1.99	4,694,564
5.0	7,122,886
12.56	9,052,743
31.55	11,151,548

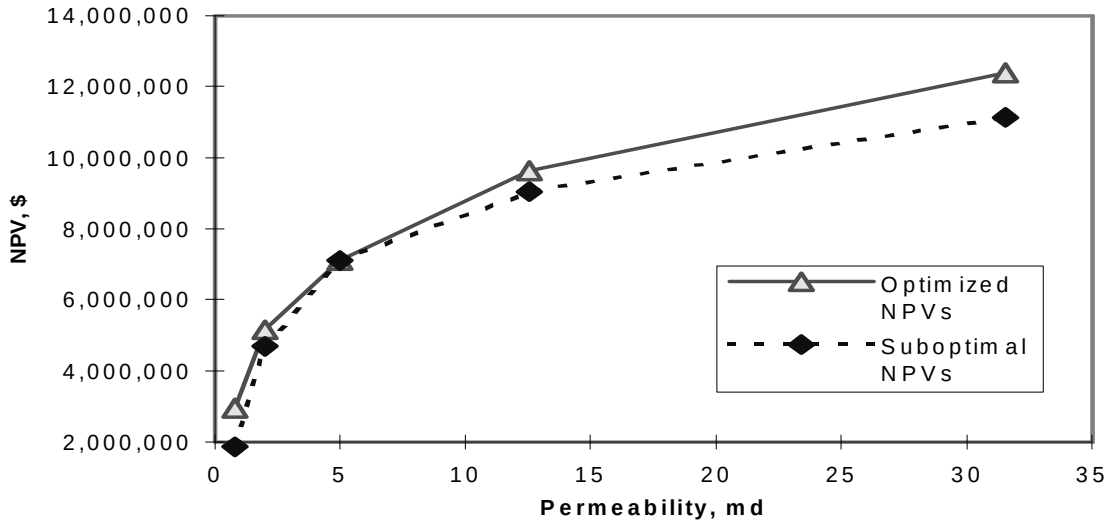

Figure 5.7: Discrepancy in NPV vs. Permeability

Table 5.10: Value of Certainty in Permeability

Representative Permeability, md	Optimal NPV	NPV tubing - 3.0"; sep pressure - 200.8 psia	Discrepancy	Probability	Cost of Uncertainty
0.792	2,957,346	1,874,792	1,082,552	5%	54,128
1.99	5,177,912	4,694,564	483,348	25%	120,837
5	7,122,886	7,122,886	0	40%	0
12.56	9,657,152	9,052,743	604,409	25%	151,102
31.55	12,394,272	11,151,548	1,242724	5%	62,136
				Sum	388,203

5.3 Porosity

Porosity is distributed as listed in Table 5.11. Five percent of porosities fall within 22.5 and 25.5 percent. A quarter of the porosities fall within 25.5 and 28.5 percent. The next 40 percent of porosities fall between 28.5 percent and 31.5 percent. Twenty-five percent of porosities fall within 31.5 and 34.5 percent. The final five percent of the porosities fall between 34.5 and 37.5 percent.

Table 5.11: Distribution of Porosities

Cumulative Distribution Function, Minimum	Cumulative Distribution Function, Minimum	Minimum Porosity, %	Maximum Porosity, %
0.0	0.05	22.5	25.5
0.05	0.3	25.5	28.5
0.3	0.7	28.5	31.5
0.7	0.95	31.5	34.5
0.95	1.0	34.5	37.5

For each of these brackets, a representative value was selected, as listed in Table 5.12. For each of the representative values, the engineering parameters were optimized. The results of these optimization runs is presented in Table 5.13. The various optima are illustrated in Figure 5.8. Figure 5.9 illustrates the optimized NPV vs. the porosity.

Table 5.12: Representative Porosities for Optimization

Cumulative Distribution Function, Minimum	Cumulative Distribution Function, Minimum	Representative Porosity %	Probability
0.0	0.05	24.0	5%
0.05	0.3	27.0	25%
0.3	0.7	30.0	40%
0.7	0.95	33.0	25%
0.95	1.0	36.0	5%

Table 5.13: Optimization Results - Porosity Distributed

Representative Porosity %	Optimum Values		Optimal NPV
	Tubing Diameter, inches	Separator Pressure, psia	
24	4.14	125.4	6,177,130
27	3.19	155.6	6,544,556
30	3	200.8	7,122,886
33	2.9	200.8	7,657,688
36	2.81	200.8	8,162,199

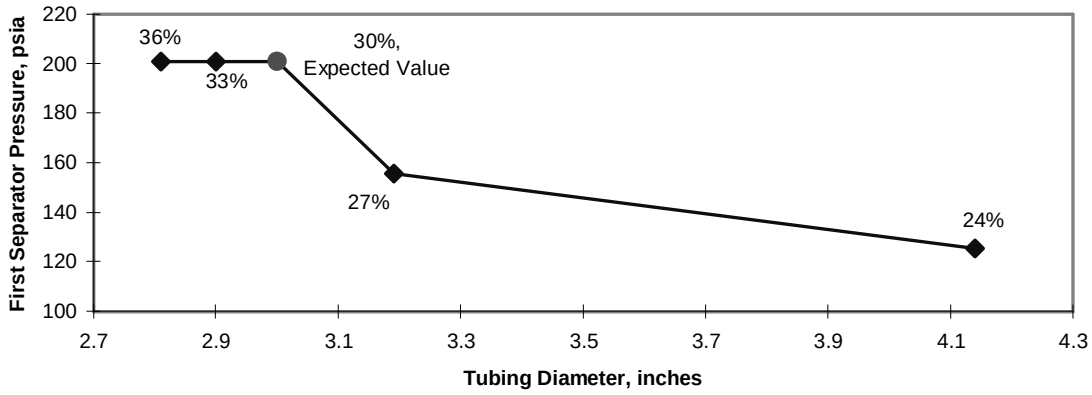

Figure 5.8: Optimal Parameter Path - Porosity

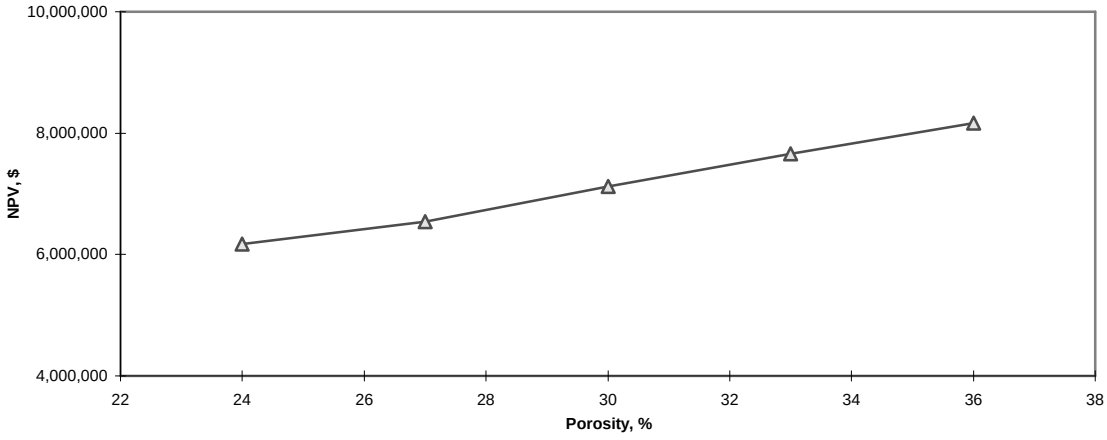

Figure 5.9: NPV vs. Porosity

The NPVs based on optimizing parameters for the expected porosity are listed in Table 5.14. The discrepancies between the suboptimal NPVs in Table 5.14 and the perfect knowledge, optimal NPVs in Table 5.13 are illustrated in Figure 5.10. The cost of uncertainty in porosity is $194,328. The calculations leading to this figure are in Table 5.15.

Table 5.14: NPV if Engineering Parameters Optimized for Expected Porosity

Representative Porosity	NPV tubing - 3.0"; sep pressure - 200.8 psia
24%	5,766,110
27%	6,088,088
30%	7,122,886
33%	7,509,493
36%	7,709,976

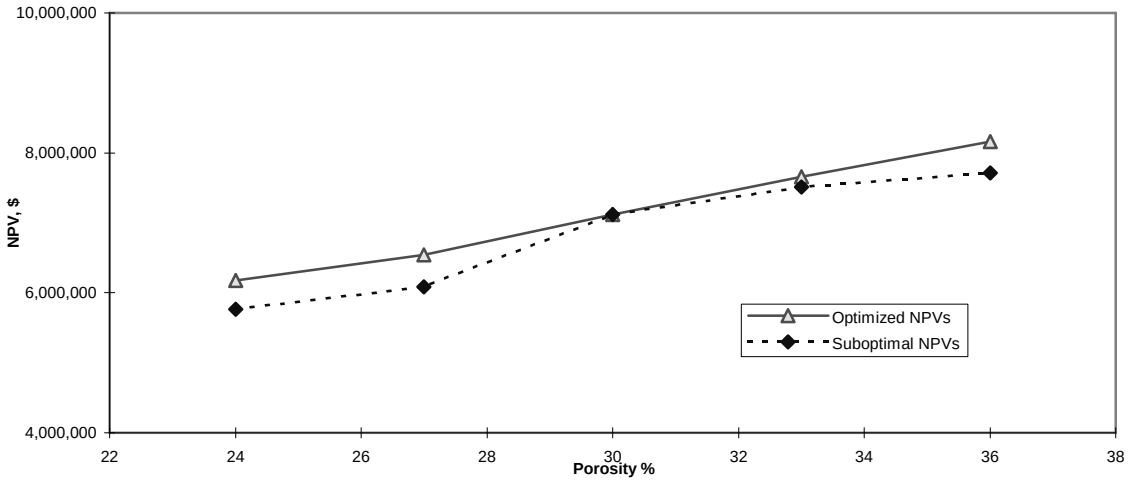

Figure 5.10: Discrepancy in NPV vs. Porosity

Table 5.15: Value of Certainty in Porosity

Representative Porosity, %	Optimal NPV	NPV tubing - 3.0"; sep pressure - 200.8 psia	Discrepancy	Probability	Cost of Uncertainty
24	6,177,130	5,766,110	411,020	5%	20,551
27	6,544,556	6,088,088	456,468	25%	114,117
30	7,122,886	7,122,886	0	40%	0
33	7,657,688	7,509,493	148,195	25%	37,049
36	8,162,199	7,709,976	452,223	5%	22,611
				Sum	194,328

5.4 Areal Extent

Table 5.16 illustrates the probability distribution for the areal extent of this reservoir. The first quintile ranges from 40 acres to 85 acres. The second quintile contains areal extents ranging from 85 to 130 acres. The third quintile ranges from 130 to 190 acres. The next 20 percent falls within the range of 190 to 255 acres. The last quintile contains areal extents from 255 to 350 acres.

Table 5.16: Distribution of Areal Extents

Cumulative Distribution Function, Minimum	Cumulative Distribution Function, Minimum	Minimum Areal Extent, acre	Maximum Areal Extent, acre
0.0	0.2	40	85
0.2	0.4	85	130
0.4	0.6	130	190
0.6	0.8	190	255
0.8	1.0	255	350

Table 5.17 lists the representative values chosen for these quintiles. The engineering parameters were optimized for each of these representative values. The results of this optimization are presented in Table 5.18. The locus of these optimized engineering

parameters is illustrated in Figure 5.11. Figure 5.12 shows the relationship between the optimized NPV and the areal extent of the reservoir.

Table 5.17: Representative Areal Extents for Optimization

Cumulative Distribution Function, Minimum	Cumulative Distribution Function, Minimum	Representative Areal Extent acres	Probability
0.0	0.2	62.5	20%
0.2	0.4	107.5	20%
0.4	0.6	160	20%
0.6	0.8	222.5	20%
0.8	1.0	302.5	20%

Table 5.18: Optimization Results - Areal Extent Distributed

Representative Areal Extent acre	Optimum Values		Optimal NPV
	Tubing Diameter, inches	Separator Pressure, psia	
62.5	1.67	170.63	3,481,980
107.5	3	155.56	5,281,408
160	3	200.79	7,122,886
222.5	2.62	215.87	8,751,967
302.5	2.81	246.03	10,693,290

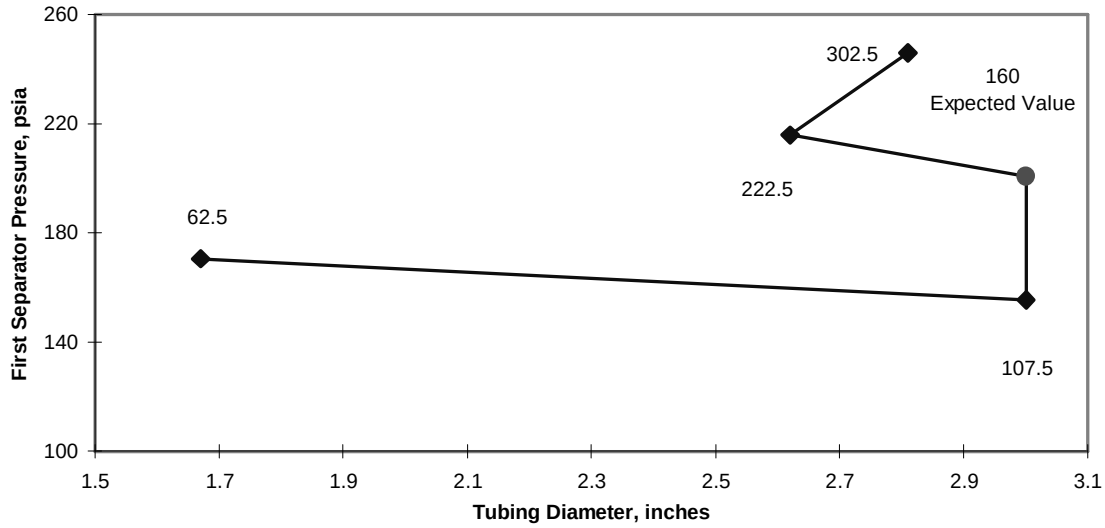

Figure 5.11: Optimal Parameter Path - Areal Extent

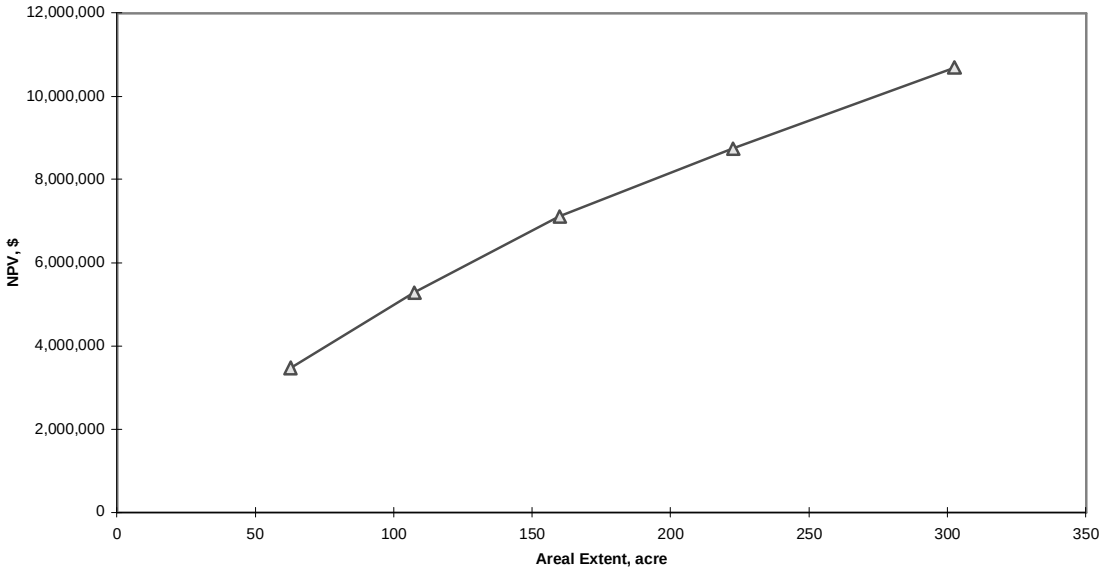

Figure 5.12: NPV vs. Areal Extent

The NPVs based on optimizing parameters for the expected porosity are listed in Table 5.19. The differences between these NPVs and the perfect knowledge NPVs from Table 5.18 are shown in Figure 5.13. The cost of uncertainty in areal extent is $165,001. The calculations leading to this figure are in Table 5.20. The cost of shooting a seismic line, or drilling an exploratory well, or any other means of exploring the areal extent of this reservoir must be measured against this cost.

Table 5.19: NPV if Engineering Parameters Optimized for Expected Areal Extent

Representative Areal Extent	NPV tubing - 3.0"; sep pressure - 200.8 psia
62.5	3,005,747
107.5	5,121,758
160	7,122,886
222.5	8,696,940
302.5	10,559,195

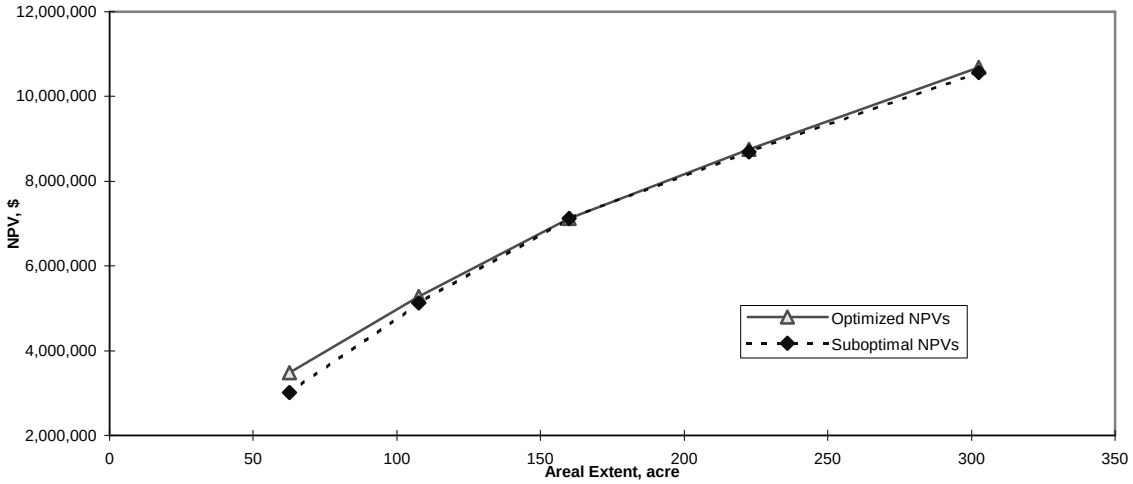

Figure 5.13: Discrepancy in NPV vs. Areal Extent

Table 5.20: Value of Certainty in Areal Extent

Representative Areal Extent, acre	Optimal NPV	NPV tubing - 3.0"; sep pressure - 200.8 psia	Discrepancy	Probability	Cost of Uncertainty
62.5	3,481,980	3,,005,747	476,233	20%	95,247
107.5	5,281,408	5,121,758	159,650	20%	31,930
160	7,122,886	7,122,886	0	20%	0
222.5	8,751,967	8,696,940	55,027	20%	11,005
302.5	10,693,290	10,559,195	134,095	20%	26,819
				Sum	165,001

5.5 Comparison Of Uncertainty Costs

We have calculated the costs of uncertainty for composition, permeability, porosity, and areal extent, based on the given probability distributions. The costs are summarized in Table 5.21. These costs serve as a means of deciding how much can be spent on transient testing, fluid testing, coring, logging, and or any other data gathering technique. With the given probability distributions, permeability clearly has the highest cost of uncertainty. This indicates that additional transient testing may be in order.

Table 5.21: Summary of Uncertainty Costs

Property	Uncertainty Cost
Composition	$157,993
Permeability	$388,203
Porosity	$194,328
Areal Extent	$165,001

6. Conclusions and Suggestions

This project has demonstrated that multivariate optimization techniques can be used to determine optimal engineering parameters for hydrocarbon production systems. Furthermore, it has been shown how these techniques can grant insight into the value of reservoir data.

6.1 Conclusions

6.1.1 Optimization techniques

- Multicomponent compositional field models can have nonsmooth objective function surfaces which may prevent gradient based methods from working.
- Polytope techniques work well for a limited number of decision variables, but polytopes should be started from a variety of positions to prevent convergence to local minima.
- For time dependent variables, a multistep polytope technique can be used to improve the convergence behavior of the polytope.
- Genetic algorithms are very robust. The test problems showed that properly tuned genetic algorithms did not need to be run for more than one random number seed.
- Modifications to the genetic algorithm, notably mutation changing and best member extrapolation, can improve genetic algorithm performance significantly.

6.1.2 Sensitivity Study

- Optimization techniques combined with a given probability distribution can provide estimates for the value of certainty in reservoir data.
- These estimates allow comparison between different data gathering options.

6.2 Suggestions

6.2.1 Optimization techniques

- Additional study into hybrid optimization techniques is needed. Genetic algorithms can be combined with polytopes and geostatistical Kriging to improve genetic algorithm performance. Other modifications, such as the Fang (1980) algorithm should be further explored.
- The complexity level of test models should be further increased to determine which techniques are best applicable for real, industrial problems.

6.2.2 Sensitivity Study

- The sensitivity study demonstrated in Section 5 should be modified to involve multivariate probability distributions. This would yield better estimates of uncertainty costs.
- In the case that reservoir parameter certainty is not going to change, it is possible to optimize engineering parameters based on a multivariate probability distribution rather than on an expected value. That is, find an optimized set of engineering parameters that does not necessarily correspond to the expected reservoir parameters.

Nomenclature

a_i	Redlich-Kwong parameter, of component i.
a_{ij}	Redlich-Kwong cross parameter, for components i and j.
a_m	Redlich-Kwong parameter, mixture Z.
A	Constant used in cubic root solution; Areal extent of reservoir, sq. ft.
A_c	Cross-sectional area of choke.
b_i	Redlich-Kwong parameter, of component i.
b_m	Redlich-Kwong parameter, mixture Z.
B	Constant used in cubic root solution.
c	Constant for scaling matrices
C	Choke discharge coefficient.
C_D	Choke discharge coefficient.
C_L	No-slip in-situ volume fraction of liquid.
D	Tubing diameter, ft.
D_2	Choke diameter, inches.
ΔL	Depth change, ft.
Δt	Time step length, days.
ΔP_H	Hydrostatic pressure change over length, psia.
ΔP_F	Frictional pressure change over length, psia.
E_L	In-situ volume fraction of liquid, considering holdup.
f_f	Fanning friction factor.
$\hat{f}_i^p$	Partial fugacity of component i in phase p.
F	Critical/subcritical boundary function; Objective function.
g	Gravitational acceleration, 32.2 ft / second ^{2}.
g_c	Gravitational constant, 32.2 lbm-ft/(lbf-second ^{2}).
$\tilde{g}$	Gradient vector.
G	Hessian matrix.
h	Reservoir thickness, ft.
HHP	Compression horsepower.
I	Identity matrix.

k	Reservoir permeability, md; Time step index; Ratio of specific heats .
k_{rp}	Relative permeability of phase p.
K	Equilibrium ratio.
K_i	Equilibrium ratio of component i in mixture Z (Y_i / X_i).
L	Liquid phase mole fraction; depth, ft.
M	Choke mass rate, lbm/day.
M_p	Phase p molecular weight, lbm/lb. mol.
n	Number of components in mixture; Polytropic exponent for gas.
N	Total reservoir mass flow rate, lb. mol/day.
N_k	Total reservoir mass, lb. mol.
$N_{k,i}$	Reservoir mass of component i at time step k, lb. mol.
n_p	Phase p relative permeability exponent.
N_p	Reservoir mass rate of phase p, lb. mol/day.
O_n	Terms order n or higher.
p	Constant for cubic root solving; Pressure of interest, psia.
$\tilde{p}$	Decision variable change vector.
$\bar{p}$	Average reservoir pressure, psia.
p_{ci}	Critical pressure of component i, psia.
p_{ri}	Reduced pressure of component i.
p_{wf}	Sandface pressure, psia.
p_{wfr}	Sandface pressure - reservoir model, psia.
p_{wft}	Sandface pressure - tubing model, psia.
q	Constant for cubic root solving; Volumetric flow rate, bbl/day.
q_p	Volumetric flow rate, bbl/day.
Q	Quadratic approximation function.
r	Constant for cubic root solving.
R	Universal gas constant.
r_e	Radius of reservoir, ft.
R_e	Reynold's number.
r_w	Radius of wellbore, ft.
S_p	Phase p saturation.
S_{pr}	Residual phase p saturation.

T	Temperature of interest, ^{o}R.
T_{ci}	Critical temperature of component i, ^{o}R.
T_{ri}	Reduced temperature of component i.
V	Gas phase mole fraction; Specific volume, cu ft / lb. mole, Specific volume, cu ft / lbm.
V_b	Bubble rise velocity, ft / second.
V_M	Mixture velocity, ft/second.
V_{Msp}	Modified superficial velocity of phase p, ft/second.
V_{sp}	Superficial velocity of phase p, ft/second.
V_t	Total gas rise velocity, ft/second.
$\tilde{x}$	Vector of decision variables.
X	Composition of liquid phase of mixture Z.
X_i	Mole fraction of component i in liquid phase X.
x_1, x_2, x_3	Solutions of cubic root.
x	Composition of liquid phase of mixture Z; Free gas quality.
x_i	Mole fraction of component i in liquid phase X.
Y	Composition of gas phase of mixture Z.
Y_i	Mole fraction of component i in gas phase Y.
y	Composition of gas phase of mixture Z; Ratio of upstream to downstream pressure.
y_c	Critical ratio of upstream to downstream pressure.
y_i	Mole fraction of component i in gas phase Y.
y_u	Applied ratio of upstream to downstream pressure.
Z	Composition of mixture, array, see Z_i.
Z_i	Mole fraction of component i in mixture Z.
Z_p	Composition of produced fluid.

Subscripts & Superscripts

1	Upstream; Separator 1.
2	Downstream; Separator 2.
3	Separator 3.
c	Critical.

F	Frictional.
g	Gas phase.
G	Gas phase.
H	Hydrostatic.
i	Component number, out of n components.
k	Time step index; Optimization step index.
L	Liquid phase.
M	Mixture; modified.
o	Oil phase.
p	Phase.
r	Reservoir.
s	Superficial.
$sepn$	Separator number n.
st	Stock tank.
t	Tubing.
T	Transpose.
u	Applied value.
wh	Wellhead.
wf	Sandface flowing.

Greek Letters

α	Constant used in cubic root solving.
β	Constant used in cubic root solving.
ε	Tolerance; Absolute pipe roughness, inches.
ϕ	Porosity.
$\hat{\phi}_i^p$	Partial fugacity constant for component i, in phase p.
μ_p	Viscosity of phase p, cp.
ω	Acentric factor.
ρ_p	Phase p density, lbm/cu ft.
σ	Interfacial surface tension between oil and gas, dyne/cm.
$\sigma_{p1,p2}$	Interfacial surface tension between phase 1 and phase 2, dyne/cm.

References

Amyx, J.W., Bass, D.M., and Whiting, R.L.: *Petroleum Reservoir Engineering*, McGraw-Hill Book Company, New York, (1960).

Aziz, K., Govier, G.W., and Fogarasi, M.: "Pressure Drop in Wells Producing Oil and Gas," *J. Canadian Petro. Tech.* (1972).

Bard, Y.: *Nonlinear Parameter Estimation*, Academic Press, London, (1974).

Carroll, J.A.: "Multivariate Production Systems Optimization," MS report, Stanford U., Stanford, CA (1990).

Colebrook, C.F., and White, C.M.: "The Reduction of Carrying Capacity of Pipes With Age," *Jour. Inst. of Civil Engrs.* (1937).

Duns, H., and Ros, N.C.J.: "Vertical Flow of Gas and Liquid Mixtures in Wells," *Proc., 6th World Petroleum Congress* (1963).

Dawkins, R., *The Blind Watchmaker*, W.W. Norton, New York, (1986).

Economides, M.J., Hill, A.D., and Ehlig-Economides, C.: *Petroleum Production Systems*, PTR Prentice Hall, Englewood Cliffs, New Jersey (1994).

Fang, K.: "Application of Number Theory in Test Design," *ACTA Mathematicae Applicatae Sinica* (November 1980) In Chinese.

Fayers, F.J.: Waterflooding Notes, unpublished class notes, Stanford University, Stanford, CA (1994).

Fortunati, F.: "Two-Phase Flow Through Wellhead Chokes", SPE 3742, presented at SPE European Meeting, (1972).

Fujii, H.: "Multivariate Production Systems Optimization in Pipeline Networks," MS report, Stanford U., Stanford, CA (1993).

Fundamentals of Petroleum, second edition, Petroleum Extension Service, Austin TX (1981).

Gill, P.E., Murray, W., and Wright, M.H.: *Practical Optimization*, Academic Press, London (1981).

Goldberg, D.E.: *Genetic Algorithms*, Addison-Wesley, Reading, MA, (1989).

Horne, R.N.: Optimization, unpublished class notes, Stanford U., Stanford, CA (1995).

McCain, W.D.: *The Properties of Petroleum Fluids*, PennWell, Tulsa, (1990).

Ravindran, N.: "Multivariate Optimization of Production Systems - the Time Dimension," MS report, Stanford U., Stanford, CA (1992).

Redlich, O., and Kwong J.N.S.: “On the Thermodynamics of Solutions: V: An Equation of State. Fugacities of Gaseous Solutions,” *Chem. Review*, 44 (1949).

Sachdeva, R., Schmidt, Z., Brill, J.P., and Blais, R.M.: “Two-Phase Flow Through Chokes”, SPE 15657, presented at SPE Annual Technical Conference and Exhibition, (1986).

Thiele, M.: Thermodynamics of Phase Behavior, unpublished class notes, Stanford University, Stanford, CA (1995).

Thompson, R.S., Wright, J.D.: *Oil Property Evaluation*, Thompson-Wright Associates, Golden CO, 1985.

Walas, S.M.: *Phase Equilibria in Chemical Engineering*, Butterworth-Heinemann, Boston, (1985).

Wattenbarger, R.A.: Gas Engineering, unpublished class notes, Texas A&M U., College Station, TX (1992).

Whitley, D.: A Genetic Algorithm Tutorial, unpublished manuscript, Colorado State U., (1995).

APPENDIX

SOURCE CODE FOR MODEL

```
************************************************************************
C   PROGRAM WRITTEN BY MILES PALKE
C   FOR MASTERS DEGREE
C   9/94 TO 8/96
c   SUPRI-D RESEARCH GROUP
C   THIS IS THE MAIN DRIVER
************************************************************************
************************************************************************
      program optimize
      parameter (maxper=20)
      real obfunc,xvariables(8,maxper)
      real constraints(8,2,maxper)
      real delta(8,maxper),gradient(2),hessian(2,2)
      integer i,j,k,numperiods,freevar(8,maxper),gabits(8,maxper)
      integer tgabits
      integer varcount,varindexi(8*maxper),varindexj(8*maxper)
      real x,y


      open (unit=27,file='decvar.in',status='old')
      open (unit=29,file='plot',status='unknown')
      open (unit=30,file='plot2',status='unknown')
      open (unit=31,file='plot3',status='unknown')
      open (unit = 55,file='rates',status='unknown')
      open (unit = 73,file='press',status ='unknown')
      open (unit = 77,file='econ',status ='unknown')


 10   format ('TIME',2x,'RATE',2x,'CUMULATIVES',2x
     + ,'REVENUE & EXPENSES',21x,'NPW')
 11   format ('Start ',' Inj  ',' Oil   Gas',3x,
     + 'Oil      Gas     Oper   Sep   Horse',
     + 4x,'total!')
 12   format ('days  ','MCFPD ','MSTB  MMSCF',
     +'  M$       M$      M$     M$     M$  '
     + 2x,'  M$  ')
      write (55,*) 'time      qosurf    qginj     qgsold'
      write (73,*) '   time      pwf     pbarres    pinj       So'
      write (77,10)
      write (77,11)
      write (77,12)
 13   format(f10.4,i10,2f10.4)

      call datain3(maxper,numperiods,xvariables,constraints,freevar,
     + delta,gabits,tgabits)
      do i = 1,8
        do j = 1,numperiods
          write (*,13) xvariables(i,j),freevar(i,j),constraints(i,1,j)
     +    ,constraints(i,2,j),delta(i,j)
         enddo
      enddo
      varcount = 0
      do i = 1,numperiods*8
         varindexi(i) = 0
         varindexj(i) = 0
      enddo
      tgabits = 0
      do i = 1,8
         do j = 1,numperiods
          if (freevar(i,j).eq.1) then
             varcount = varcount + 1
             tgabits = tgabits+gabits(i,j)
             varindexi(varcount) = i
             varindexj(varcount) = j
          endif
         enddo
      enddo
      print *, 'varcount =',varcount

C  AT THIS POINT WE CALL THE OPTIMIZATION ROUTINE OF INTEREST

       call genetic(varcount,varindexi,varindexj,numperiods,
     + xvariables,constraints,delta,freevar,gabits,tgabits)

C  NEXT TWO POSSIBILLITIES ARE MARQUARD AND POLYTOPE(PALKETOPE)
C  OPTIMIZATION ROUTINES

c       call palketope(varcount,varindexi,varindexj,numperiods,
c     + xvariables,constraints,delta,freevar)
c       call modmarq(varcount,varindexi,varindexj,numperiods,
c     + xvariables,constraints,delta,freevar)

      end
************************************************************************
************************************************************************
C    THIS ROUTINE PERFORMS THE MODIFIED MARQUARDT OPTIMIZATION
C    THE TECHNIQUE IS DESCRIBED IN THE REPORT
************************************************************************
      subroutine modmarq(varcount,varindexi,varindexj,numperiods,
     + xvariables,constraints,delta,freevar)
      real obfunc,xvariables(8,numperiods)
      real constraints(8,2,numperiods)
      real delta(8,numperiods)
      integer numperiods,freevar(8,numperiods)
      integer varcount,varindexi(varcount),varindexj(varcount)
      real gradient(varcount)
      real hessian(varcount,varcount)
      real gscale(varcount)
      real hscale(varcount,varcount)
      real junkhess(varcount,varcount)
      real junkhess2(varcount,varcount)
      real ipvt(varcount)
      real cond,mu
      real work(varcount),z(varcount),x(varcount)
      real rowin,rowout,xscaled(varcount),gamma
      real lastobv,lastx(8,numperiods)
      real pchangeobv,pchangex
      integer i,j,k,check
************************************************************************
      obfunc = 99999.
 10   call hsandgrad(varcount,varindexi,varindexj,numperiods,
     + xvariables,constraints,delta,freevar,obfunc,gradient,hessian)
      lastobv = obfunc
      do i = 1,8
         do j = 1,numperiods
            lastx(i,j) = xvariables(i,j)
         enddo
      enddo
      check = 0
      mu = .01
      write (29,*) (xvariables(varindexi(j),varindexj(j)),j=1,varcount)
     + ,obfunc
 12   print *, 'mu =',mu
      do i = 1,varcount
        hessian(i,i) = hessian(i,i)+mu
      enddo
      print *, (xvariables(varindexi(j),varindexj(j)),j=1,varcount)
     + ,obfunc
      do i = 1,varcount
          print *, 'i,gradient',i,gradient(i)
```

```
      enddo
      print *, 'hessian ='
      do i = 1,varcount
        print *, (hessian(i,j),j=1,varcount)
      enddo
      determinant = 1.0
      do i = 1,varcount
         if(hessian(i,i).lt.0.0) determinant = -1.0
      enddo
      do i = 1,varcount
         do j = 1,varcount
           junkhess(i,j) = hessian(i,j)
         enddo
      enddo
      call decomp(varcount,varcount,junkhess,
     +  cond,ipvt,work,junkhess2,z)
      call solve(varcount,varcount,junkhess,gradient,ipvt,x)
      print *, 'cond =',cond
      print *, 'x =',x
      rowin = 0.0
      do i = 1,varcount
         rowin = rowin + x(i)**2
      enddo
      rowin = sqrt(rowin)
      do i = 1,varcount
       xscaled(i) = -x(i)/rowin
       print *, 'xscaled(i) =',xscaled(i)
      enddo
*********************************************************************
C   GAMMA REPRESENTS THE RATE OF CHANGE ALONG THE NEWTON STEP
C   IF THE STEP IS NOT IN DESCENT, MU IS INCREASED AND THE NEWTON
C   STEP IS RECALCULATED.
*********************************************************************
      gamma = 0.0
      do i = 1,varcount
        gamma = gamma+xscaled(i)*gradient(i)
      enddo
      print *, 'gamma',gamma
      if (gamma.gt.0.0.or.determinant.lt.0.0) then
      print *, 'must modify mu'
      check = 1
      do i = 1,varcount
        hessian(i,i) = hessian(i,i)-mu
      enddo
      mu = mu*10.
      goto 12
      endif
*********************************************************************
      if (check.eq.0) mu = mu / 10.
      print *, 'rowin = ',rowin
      call search2(varcount,varindexi,varindexj,numperiods,
     + xvariables,constraints,delta,xscaled,rowin,rowout,obfunc,
     +  gamma)
      print *, 'rowout =',rowout
      do i = 1,varcount
         xvariables(varindexi(i),varindexj(i)) =
     +    xvariables(varindexi(i),varindexj(i)) + rowout*xscaled(i)
         print *, xvariables(varindexi(i),varindexj(i))
      enddo
      print *, 'after search2 obfunc =',obfunc
      pchangeobv = 100.*(obfunc-lastobv)/obfunc
      print *, 'percent change obv =',pchangeobv
      if (abs(pchangeobv).gt.0.01) goto 10
      pchangex = 0.0
      do i = 1,varcount
       do j = 1,numperiods
         pchangex = 100.*(xvariables(i,j)-lastx(i,j))/xvariables(i,j)
         print *, 'i,j, change decvar',i,j,pchangex
         if (abs(pchangex).gt.0.01) goto 10
       enddo
      enddo
      print *, 'routine has converged!'
      return
      end
*********************************************************************
*********************************************************************
C  THIS ROUTINE CALCULATES THE HESSIAN MATRIX AND THE GRADIENT FOR THE
C  MODEL AT THE GIVEN POINT ON THE MODEL'S OBJECTIVE FUNCTION SURFACE
*********************************************************************
      subroutine hsandgrad(varcount,varindexi,varindexj,numperiods,
     + xvariables,constraints,delta,freevar,obout,gradient,hessian)
      real obfunc,xvariables(8,numperiods)
      real constraints(8,2,numperiods)
      real delta(8,numperiods)
      integer numperiods,freevar(8,numperiods)
      integer varcount,varindexi(varcount),varindexj(varcount)
      real gradient(varcount),obval(varcount+1,2)
      real obcross(varcount,varcount)
      real hessian(varcount,varcount)
      real obout
      integer i,j,k
*********************************************************************
      row = 100.
        obval(1,1) = obout
      if (obout.eq.99999.) then
        call caller(numperiods,xvariables,obfunc)
        obval(1,1) = -obfunc
        obout = -obfunc
      endif
  5   do i = 1,varcount
         xvariables(varindexi(i),varindexj(i)) =
     +           xvariables(varindexi(i),varindexj(i)) +
     +           delta(varindexi(i),varindexj(i))
         call caller(numperiods,xvariables,obfunc)
         obval(i+1,2) = -obfunc
       if (i.lt.varcount) then
       do j = i+1,varcount
                 xvariables(varindexi(j),varindexj(j)) =
     +           xvariables(varindexi(j),varindexj(j)) +
     +           delta(varindexi(j),varindexj(j))
         call caller(numperiods,xvariables,obfunc)
         obcross(i,j) = -obfunc
                 xvariables(varindexi(j),varindexj(j)) =
     +           xvariables(varindexi(j),varindexj(j)) -
     +           delta(varindexi(j),varindexj(j))
       enddo
       endif
         xvariables(varindexi(i),varindexj(i)) =
     +           xvariables(varindexi(i),varindexj(i)) -
     +           2.*delta(varindexi(i),varindexj(i))
         call caller(numperiods,xvariables,obfunc)
         obval(i+1,1) = -obfunc
         xvariables(varindexi(i),varindexj(i)) =
     +           xvariables(varindexi(i),varindexj(i)) +
     +           delta(varindexi(i),varindexj(i))
       enddo
       do i = 1,varcount
          gradient(i) = 0.5*(obval(i+1,2)-obval(i+1,1))
     +         /delta(varindexi(i),varindexj(i))
          hessian(i,i) = obval(i+1,2)+obval(i+1,1)-2.*obval(1,1)
```

```
         hessian(i,i) = hessian(i,i)/delta(varindexi(i),varindexj(i))
         hessian(i,i) = hessian(i,i)/delta(varindexi(i),varindexj(i))
       if (i.lt.varcount) then
       do j = i+1,varcount
        hessian(i,j) = obcross(i,j)+obval(1,1)-obval(i+1,2)-obval(j+1,2)
        hessian(i,j) = hessian(i,j)/delta(varindexi(i),varindexj(i))
        hessian(i,j) = hessian(i,j)/delta(varindexi(j),varindexj(j))
        hessian(j,i) = hessian(i,j)
       enddo
       endif
       enddo
       do i = 1,varcount
           print *, 'i,gradient',i,gradient(i)
       enddo
       print *, 'hessian ='
       do i = 1,varcount
         print *, (hessian(i,j),j=1,varcount)
       enddo
       return
       end
**********************************************************************
**********************************************************************
C  THIS IS A VERSION OF THE LINE SEARCH DESCRIBED BY BARD.
C  THE SEARCH MOVES ALONG THE DIRECTION INDICATED BY XSCALED TO FIND
C  A POSITION BETTER THAN THE STARTING POINT.  FIRST IT TRIES THE
C  NEWTON STEP
**********************************************************************
       subroutine search2(varcount,varindexi,varindexj,numperiods,
      + xvariables,constraints,delta,xscaled,rowin,rowout,obin,
      + gamma)
       real obfunc,xvariables(8,numperiods)
       real constraints(8,2,numperiods)
       real delta(8,numperiods)
       integer numperiods,freevar(8,numperiods)
       integer varcount,varindexi(varcount),varindexj(varcount)
       real gradient(varcount)
       real xscaled(varcount),gamma,rowstow,obstow
       real modulus,rowin,rowout,rowmax,rowmin,rownow
       real xplay(8,numperiods),obin,a,b,c
       integer i,check,check2,count
**********************************************************************
       rowmax = 100000000.0
       rowmin = -100000000.0
       do i = 1,8
        do j = 1,numperiods
         xplay(i,j) = xvariables(i,j)
        enddo
       enddo
       check = 0
       check2 = 0
       count = 0
       do i = 1,varcount
          if (xscaled(i).eq.0.0) xscaled(i) = 0.00000001
       enddo
       do i = 1,varcount
         print *, 'xscaled(i) =',xscaled(i)
         rowtemp = (constraints(varindexi(i),1,varindexj(i))
     +    + delta(varindexi(i),varindexj(i)) -
     +    xvariables(varindexi(i),varindexj(i)))/xscaled(i)
         print *, 'i,rowtemp',i,rowtemp
         if (rowtemp.gt.0.0) rowmax = min(rowmax,rowtemp)
         if (rowtemp.lt.0.0) rowmin = max(rowmin,rowtemp)
         rowtemp = (constraints(varindexi(i),2,varindexj(i)) -
     +    delta(varindexi(i),varindexj(i)) -
     +    xvariables(varindexi(i),varindexj(i)))/xscaled(i)
         print *, 'i,rowtemp',i,rowtemp
         if (rowtemp.gt.0.0) rowmax = min(rowmax,rowtemp)
         if (rowtemp.lt.0.0) rowmin = max(rowmin,rowtemp)
       enddo
       print *, 'rowmax,rowmin',rowmax,rowmin
         rownow = min(.95*rowmax,rowin)
       obstow = 10000000.
 10    do j = 1,varcount
         xplay(varindexi(j),varindexj(j)) =
     +      xvariables(varindexi(j),varindexj(j)) +
     +      rownow*xscaled(j)
         print *, 'i,xplay(i)',i,xplay(varindexi(j),varindexj(j))
         enddo
         call caller(numperiods,xplay,obfunc)
         obfunc = -obfunc
         count = count + 1
         print *, 'rownow,obfunc',rownow,obfunc
         if (obfunc.gt.obin.and.check2.eq.0) then
         a = obin
         b = gamma
         c = obfunc - a -b*rownow
         c = c/(rownow**2)
         print *, 'a,b,c=',a,b,c
         rownow = -b/(2.*c)
         print *, 'interpolated to rownow =',rownow
         check = 1
         count = count + 1
         if (count.gt.9) then
            goto 50
         endif
         goto  10
         endif
         if (obfunc.le.obin.and.check.eq.0) then
         check2 = 1
         check = 1
         rowstow = rownow
         obstow = obfunc
          rownow = min(2.*rownow,rownow+.75*(rowmax-rownow))
          goto 10
         endif
 50    rowout = rownow
       obin = obfunc
       if (obstow.lt.obfunc) then
          obin = obstow
          rowout = rowstow
       endif
       return
       end
**********************************************************************
**********************************************************************
C  CALLER CONVERST THE DECISCION VARIABLE VECTOR (xvariables) TO THE
C  CORRECT FORMAT FOR SUBROUTINE PROG. IT THEN CALLS PROG
**********************************************************************
       subroutine caller(numperiods,xvariables,obfunc)
       real diam(numperiods),depth(numperiods),rate(numperiods)
       real choke(numperiods),line(numperiods),spress(3,numperiods)
       real obfunc,xvariables(8,numperiods)
       real constraints(8,2,numperiods)
       integer i,j,k,numperiods,freevar(8,numperiods)
**********************************************************************
        do j = 1,numperiods
         diam(j) = xvariables(1,j)
         depth(j) = xvariables(2,j)
         rate(j) = xvariables(3,j)
         choke(j) = xvariables(4,j)
```

```
        line(j) = xvariables(5,j)
        spress(1,j) = xvariables(6,j)
        spress(2,j) = spress(1,j)*xvariables(7,j)
        spress(3,j) = spress(1,j)*xvariables(8,j)
      enddo
      call prog(diam,depth,rate,choke,line,spress,
     +     obfunc,numperiods)
      return
      end
*********************************************************************
*********************************************************************
C  THIS ROUTINE READS IN THE INFORMATION ABOUT THE DECISCION VARIABLES
C  THE INPUT IS DECVAR.IN  IT READS THE STARTING VALUES (DOES NOT MATTER
C  FOR GA), IF THE VARIABLE IS VARIABLE OF A CONSTANT, MINIMUM VALUE
C  MAXIMUM VALUE, DELTA SIZE (FOR HESSIAN AND GRADIENT, OR POLYTOPE STEP)
C  AND NUMBER OF BITS FOR GA.  FIRST IT READS IN THE NUMBER OF TIME-PERIODS,
C  AND A CHECK TO SEE IF THE FIRST LINE IS USED FOR ALL TIME PERIODS (IF
C  IT IS, DON'T INCLUDE OTHER LINES)
*********************************************************************
      subroutine datain3(max,numperiods,xvariables,constraints,freevar,
     + delta,gabits,tgabits)
      real xvariables(8,max)
      real constraints(8,2,max)
      real delta(8,max)
      integer i,j,numperiods,checker,freevar(8,max),max
      integer gabits(8,max),tgabits
*********************************************************************
  5   format(2i10)
  6   format(f10.4,i10,2f10.4,f10.6)
      read (27,*)
      read (27,5) numperiods,checker
      read (27,*)
      read (27,*)
      do i = 1,8
        read (27,*)
        read (27,*) xvariables(i,1),freevar(i,1),constraints(i,1,1),
     +              constraints(i,2,1),delta(i,1),gabits(i,1)
        if (freevar(i,1).eq.0) then
           constraints(i,1,1) = xvariables(i,1)
           constraints(i,2,1) = xvariables(i,1)
        endif
        if (numperiods.gt.1) then
          if(checker.eq.1) then
          do j = 2,numperiods
              xvariables(i,j) = xvariables(i,1)
              freevar(i,j) = freevar(i,1)
              constraints(i,1,j) = constraints(i,1,1)
              constraints(i,2,j) = constraints(i,2,1)
              delta(i,j) = delta(i,1)
              gabits(i,j)=gabits(i,1)
          enddo
         else
          do j = 2,numperiods
           read (27,*) xvariables(i,j),freevar(i,j),constraints(i,1,j),
     +          constraints(i,2,j),delta(i,j),gabits(i,1)
        if (freevar(i,j).eq.0) then
           constraints(i,1,j) = xvariables(i,1)
           constraints(i,2,j) = xvariables(i,1)
        endif
          enddo
         endif
        endif
      enddo
      return
      end
*********************************************************************
*********************************************************************
C   THIS IS THE MAIN MODEL DRIVING PROGRAM.  IT PERFORMS THE SIMULATIONS
C   REQUESTED BY THE OPTIMIZATION ROUTINES.
*********************************************************************
      subroutine prog(d,injdepth,ngasinj,chokedm,surfdiam,
     +   psep,npw,numperiods)
*********************************************************************
      parameter (np = 8)
      real zinput(np),pc(np),tc(np),omega(np),vstand(np),m(np),moody
      real xs(np),ys(np),ai(np),bi(np),fl(np),fv(np)
      real r,aij(np*np),k(np),qo,qg
      real epsilon,d(numperiods),t2,length,ptop,ttop,tbottom
      real error,mx2,zinj(np),injdepth(numperiods),injrate,injgmw
      real l,a,b,vl2,ngasinj(numperiods),mx1o,mx2o
      real ctepsilon,vlvpres,dc,pcbottom,pcwh,endtime
      real psep(3,numperiods),tsep(3),noil,ngas
      real areal,perm,pinit,porosity,thick,restemp,re,mbar(np)
      real swc,sor,sgr,mixdens,pwfres,zout(np)
      real krof,krgf,so,resvolume,resmass,pbar,time,mpro(np)
      real deltime,zres(np),pbo1,pbo2,pbarnew
      real steplength,delpmax,stepconv,rtconvg,outflcon,inflcon
      real pwfconv,resupconv,ztol,mxs,mxmax,mxmin,pbarerr
      real chokecoef,chokeconv
      integer n,v2,v3,period,numperiods
      real oilprice,oilesc,oilterm
      real gasprice,gasesc,gasterm
      real opcost,opesc,opterm
      real sepcost,sepesc,septerm
      real horseprice,horseesc,horseterm
      real discrate
      real npw,oilcum,gascum,peroil,pergas,perop,persep,perhorse
      real ngtemp,dtemp,idtemp
      real chokedm(numperiods),surfdiam(numperiods)
      real maxpress
      real dummy,dummy2(4),dummy3(3),pseptemp(3)
*********************************************************************
      npw = 0.0
      time = 0.*365./24.0
      deltime = 365./2.
      endtime = 365.*20.
      period = 1
  1   format (4f15.5)
  2   format (10f10.2,f10.6,f10.4,f10.2)
  3   format (f10.2,f10.4,f10.7)
 13   format (4(x,f5.1),x,f7.2,4(x,f6.2),f10.2)
 17   format (7f10.2,f20.6,f10.4,f10.2)
      oilcum = 0.0
      gascum = 0.0
      n = np
      r = 10.73
      mx2 = 0.0
*********************************************************************
      call datain(n,zinput,pc,tc,omega,tsep,v2,v3,vstand,m,
     +       length,ttop,
     +       tbottom,epsilon,ctepsilon,
     +       vlvpres,dc,areal,thick,porosity,pinit,restemp,
     +       perm,swc,sor,sgr,noil,ngas,
     +       steplength,delpmax,stepconv,rtconvg,outflcon,inflcon,
     +       pwfconv,resupconv,ztol,pbarerr,
     +       chokecoef,chokeconv,maxpress)
      call datain2(oilprice,oilesc,oilterm,gasprice,gasesc,gasterm,
     + opcost,opesc,opterm,sepcost,sepesc,septerm,horseprice,horseesc,
     + horseterm,discrate)
*********************************************************************
```

```
      do i = 1,n
          zres(i) = zinput(i)
      enddo
      call resprep(areal,re,resvolume,pinit,mixdens,mpro,mbar,
     +     resmass,n,zres,restemp,pc,tc,omega,a,b,k,fl,fv,r,
     +     ai,bi,aij,porosity,thick,m,vstand,pbar,outflcon,inflcon)
      pwfres = 0.9*pinit
      do i = 2,n
        zinj(i) = 0.0
      enddo
      zinj(1) = 1.0
      pbo1 = pinit
      pbo2 = .95*pinit
      oilcum = 0.0
      gascum = 0.0
      peroil = 0.0
      pergas = 0.0
      perop = 0.0
      persep = 0.0
      perhorse = 0.0
      ptop = 1000.
      write (77,13) time-deltime,ngasinj(period),oilcum/1000.,
     + gascum/1000.,
     + peroil/1000.,pergas/1000.,perop/1000.,persep/1000.,
     + perhorse/1000.,npw/1000.
4     mx2o = mx2
      period = int(real(numperiods)*(time+.5*deltime)/(endtime))+1
      ngtemp = ngasinj(period)
      dtemp = d(period)
      idtemp = injdepth(period)
      if (time.ge.2.*deltime) mx2 = mx2 + (mx2-mx1o)
      pbarnew = pbar + (pbo2-pbo1)
      if (time.lt.2.*deltime.and.time.gt.deltime)
     +  pbarnew = pbar +(pbar-pinit)
      mx1o = mx2o
      do i = 1,3
         pseptemp(i) = psep(i,period)
      enddo
      call outloop(pbarnew,ngasinj(period),injgmw,mxs,mx2,n,
     +     zres,zinj,injrate,
     +     ptop,ttop,d(period),pc,tc,
     +     omega,k,fl,fv,ai,bi,aij,vstand,m,injdepth(period)
     +     ,epsilon,
     +     pcbottom,vlvpres,pwfres,pwftub,length,error,a,b,t2,tbottom,
     +     zout,restemp,pbar,re,dc,thick,perm,noil,ngas,swc,sor,
     +     sgr,so,pseptemp,tsep,outflcon,inflcon,
     +     rtconvg,steplength,delpmax,pwfconv,stepconv,mxmin,mxmax,
     +     qo,qg,deltime,ztol,time,ctepsilon,pcwh,resmass,
     +     mpro,mbar,resvolume,pbarerr,chokedm(period),
     +     surfdiam(period),
     +     chokecoef,chokeconv,maxpress,resupconv)
      call econ(npw,oilcum,gascum,oilprice,oilesc,oilterm,
     + gasprice,gasesc,gasterm,opcost,opesc,opterm,
     + sepcost,sepesc,septerm,horseprice,horseesc,horseterm,
     + discrate,ngtemp,qo,qg,time,deltime,pcwh,psep(3,period),
     + peroil,pergas,perop,persep,perhorse)

      write (55,2) time-deltime,qo,ngasinj(period),qg,injdepth(period)
     +     ,gascum/1000.,npw/1000.
      write (73,17) time-deltime,pwfres,pbarnew,pcwh,mxmin,mxmax,error,
     + resmass
      write (77,13) time-deltime,ngasinj(period),oilcum/1000.,
     + gascum/1000.,
     + peroil/1000.,pergas/1000.,perop/1000.,persep/1000.,
     + perhorse/1000.,npw/1000.

      pbar = pbarnew
      pbo1 = pbo2
      pbo2 = pbar
      if (time.lt.endtime) goto 4
      return

      end
*****************************************************************
*****************************************************************
C  THIS ROUTINE PERFORMS THE PRESSURE TRAVERSE.  THERE IS ANOTHER ROUTINE
C  CALLED TRAVERSE2 WHICH IS SIMILAR, BUT TRAVERSES UPWARD RATHER THAN
C  DOWNWARD -- THIS IS A DOWNWARD TRAVERSE
*****************************************************************
      subroutine traverse(zinput,x,y,mx,my,ptop,ttop,pbottom,tbottom,d
     + ,pc,tc,omega,
     + k,fl,fv,ai,bi,aij,n,vstand,startdep,length,m,epsilon,
     + mx2,outflcon,inflcon,steplength,delpmax,stepconv,
     + halt,maxpress)
      real vstand(n),dl
      real x(n),y(n),zinput(n),pc(n),tc(n),omega(n),fl(n),fv(n),ai(n)
      real bi(n),aij(n*n),mx,my,d,p1,p2,t1,t2,depth
      real m(n),dpres,mx2
      real k(n),ptop,ttop,pbottom,tbottom,length,dpmax
      real epsilon,l,startdep,outflcon,inflcon,steplength,delpmax
      real p1o,p2o,stepconv,maxpress
      integer n,regime,halt
*****************************************************************
      halt = 0
      dpmax = delpmax
      dl = steplength
      depth = startdep
      p1 = ptop
      t1 = ttop
      p2o = p1
      p1o = p1
      if (length-depth.eq.0.0) then
         pbottom = ptop
         return
      endif
  10  depth = depth + dl
      depth = min(depth,length)
      p2 = 3.*p1 - 3.*p1o + p2o
      t1 = ttop + (tbottom-ttop)*(depth-dl-startdep)/
     +   (length-startdep)
      t2 = ttop + (tbottom-ttop)*(depth-startdep)/(length-startdep)
      call stepper(zinput,x,y,mx,my,p1,t1,p2,t2,d,pc,tc,omega,
     + k,fl,fv,ai,bi,aij,n,vstand,dl,m,epsilon,regime,l,
     + outflcon,inflcon,stepconv,maxpress)
      dpres = abs(p2-p1)
      if (dpres.gt.dpmax) dl = dl *dpmax/dpres*0.9
      if (depth+dl.ge.length) dl = length-depth
      p2o = p1o
      p1o = p1
      p1 = p2
      if (p2.ge.maxpress) then
         halt = 1
         pbottom = p2
         return
      endif
      if (depth.lt.length) goto 10
      pbottom = p2
      return
      end
*****************************************************************
```

```
**********************************************************************
C  THIS TRAVERSE IS FOR THE INJECTION GAS ONLY.  THIS TRAVERSE IS UPWARD
C  FROM THE POINT OF INJECTION
**********************************************************************
      subroutine traverse2(zinput,x,y,mx,my,pbottom,tbottom,ptop,ttop,d
     + ,pc,tc,omega,
     + k,fl,fv,ai,bi,aij,n,vstand,startdep,enddep,m,epsilon,time,
     + outflcon,inflcon,stepconv,maxpress)
      real vstand(n),dl
      real x(n),y(n),zinput(n),pc(n),tc(n),omega(n),fl(n),fv(n),ai(n)
      real bi(n),aij(n*n),mx,my,d,p1,p2,t1,t2,depth
      real m(n),dpres,enddep
      real k(n),ptop,ttop,pbottom,tbottom,length,dpmax
      real epsilon,l,startdep
      real p1o,p2o,time,stepconv,inflcon
      integer n,dsteps,regime
**********************************************************************
      dsteps = 300.
      dpmax = 20.0
      dl = (enddep-startdep)/real(dsteps)
      depth = startdep
      p1 = pbottom
      p2o = p1
      p1o = p1
      if (enddep-startdep.eq.0.0) then
         ptop = pbottom
         return
      endif
  10  depth = depth + dl
      depth = max(depth,enddep)
      p2 = 3.*p1 -3.*p1o + p2o
      t1 = tbottom +(ttop-tbottom)*(-depth+dl+startdep)/
     +   (startdep-enddep)
      t2 = tbottom +(-tbottom+ttop)*(-depth+startdep)/(startdep-enddep)
      call stepper(zinput,x,y,mx,my,p1,t1,p2,t2,d,pc,tc,omega,
     + k,fl,fv,ai,bi,aij,n,vstand,dl,m,epsilon,regime,l,outflcon,
     + inflcon,stepconv,maxpress)
      dpres = abs(p2-p1)
      if (dpres.gt.dpmax) dl = dl *dpmax/dpres*0.9
      if (depth+dl.le.enddep) dl = enddep-depth
      p2o = p1o
      p1o = p1
      p1 = p2
      if (depth.gt.enddep) goto 10
      ptop = p2
      return
      end
**********************************************************************
**********************************************************************
C  THIS SUBROUTINE PERFORMS THE ACTUAL STEPS ON THE PRESSURE TRAVERSE.
C  IT IS CALLED BY BOTH TRAVERSE AND TRAVERSE2.  SOME OF THE UNITS ARE
C  DESCRIBED BELOW
c     1 starts with cell pressure & temp.
c     flashes at the average with next pressure, etc.  calls new properties
c     based on this flash.  then figures frictional and hydrostatic based
c     on this average flash.  when this is done, it figures out pressure 2
c     and iterates around until p2 = p2old
c     all densities in lbm/ft^3
c     diameter in in
c     dl in feet
c     moody friction
c     mass flow in lbmol/day
c     velocities in ft/sec
c     go for it
**********************************************************************
      subroutine stepper(zinput,x,y,mx,my,p1,t1,p2,t2,d,pc,tc,omega,
     + k,fl,fv,ai,bi,aij,n,vstand,dl,m,epsilon,regime,l,
     + outflcon,inflcon,stepconv,maxpress)
      real moody,mwap,vstand(n),r,odens,gdens,ovisc,gvisc,dl,pold
      real x(n),y(n),zinput(n),pc(n),tc(n),omega(n),fl(n),fv(n),ai(n)
      real bi(n),aij(n*n),mx,my,p1,p2,t1,t2,d,nre,stream,pavg,tavg
      real l,vm,vls,vgs,vliquid,vvapor,mliq,mvap,m(n),dp2error,dp2
      real mtotal,mwmix,csa,densmix,k(n),vsg2,vsg3,dphyd,dpfric
      real epsilon,fmoody,dpf1,dpf2,dpftot,a,b,outflcon,inflcon
      real stepconv,maxpress,eg,el,sigma
      integer n,regime,resetks,fluid,itnum
**********************************************************************
      dp2error = stepconv
      itnum = 0
      fmoody = 0.01
      sigma = 95.
      csa = 0.00545415*d*d
      mtotal = mx+my
      r = 10.73
      tavg = (t1+t2)*0.5
      resetks = 1
 10   pavg = (p1+p2)*0.5
      itnum = itnum + 1
      pold = p2
      odens = 0.0
      gdens = 0.0
      vgs = 0.0
      vls = 0.0
      call master(l,n,zinput,x,y,pavg,tavg,pc,tc,omega,a,b,k,fl,
     +   fv,r,ai,bi,aij,resetks,vliquid,vvapor,outflcon,inflcon)
      mx = l*mtotal
      my = (1.-l)*mtotal
      mwmix = mwap(n,zinput,m)
      if (l.lt.1.0) then
         mvap = mwap(n,y,m)
         gdens = (mvap/vvapor)
         call viscman(n,pavg,tavg,y,gvisc,m,tc,pc,0,gdens)
         vgs = (my*mvap/gdens)/(86400.*csa)
      endif
      if (l.gt.0.0) then
         mliq = mwap(n,x,m)
         call oildens(n,vstand,x,pavg,tavg,m,odens)
         call viscman(n,pavg,tavg,x,ovisc,m,tc,pc,1,odens)
         vls = (mx*mliq/odens)/(86400.*csa)
      endif
      if (l.lt.0.00000001) then
         nre  = 1.488121*gdens*vgs*d/gvisc
         fmoody = moody(epsilon,d,nre,fmoody)
         dphyd = gdens*dl/144.
         regime = 0
         densmix = l*odens+(1.-l)*gdens
         el = 0.0
         vm = (mtotal*mwmix/densmix)/(86400.*csa)
         goto 20
      endif
      densmix = l*odens+(1.-l)*gdens
      vm =  vgs + vls
      call mregime(vgs,vls,gdens,odens,regime,vsg2,vsg3)
C  THESE ARE THE FLOW REGIMES RETURNED BY MREGIME
c          0 = bubble flow
c          1 = slug flow
c          2 = froth flow
c          3 = annular mist flow
       el = 1.0
       if (l.lt.1.0.and.l.gt.0.0)
```

```
     +    el = vls/vm
      if (regime.eq.0)
     + call elbubble(el,vm,vgs,sigma,gdens,odens)
      if (regime.eq.1)
     + call elslug(el,vm,vgs,sigma,gdens,odens)
      dphyd = (odens*el+gdens*(1.-el))*dl/144.
      nre = 124.0135*d*vm*odens/ovisc
      fmoody =  moody(epsilon,d,nre,fmoody)
 20   if (regime.eq.0) dpfric = 0.001295*fmoody*vm*vm*odens*dl/d
      if (regime.eq.1) dpfric = 0.001295*fmoody*vm*vm*odens*dl*el/d
      if (regime.eq.3) then
      nre = 124.0135*d*vgs*gdens/gvisc
      fmoody =  moody(epsilon,d,nre,fmoody)
      dpfric = 0.001295*fmoody*vgs*vgs*gdens*dl/d
      endif
      if (regime.eq.2) then
      dpf1 = 0.001295*fmoody*vm*vm*odens*dl*el/d
      nre = 124.0135*d*vgs*gdens/gvisc
      fmoody =  moody(epsilon,d,nre,fmoody)
      dpf2 = 0.001295*fmoody*vgs*vgs*gdens*dl/d
      dpfric = (dpf2-dpf1)*(vgs-vsg2)/(vsg3-vsg2)+dpf1
      endif
      dptot = dphyd+dpfric*dl/abs(dl)
      p2 = p1 + dptot
      p2 = min(p2,maxpress)
      dp2 = abs(p2-pold)
      if (dp2.ge.dp2error.and.itnum.lt.100) goto 10
      return
      end
*****************************************************************
*****************************************************************
C   THIS FUNCTION RETURNS THE MOODY FRICTION FACTOR AFTER AN ITERATIVE
C   PROCEDURE - THE PROCEDURE USES NEWTONS ROOT FINDING METHOD
*****************************************************************
      real function moody(epsilon,diameter,Nre,moodyold)
      real epsilon,diameter,Nre,moodyold,moody2,err1,err2,derr
      integer i
*****************************************************************
      moody = moodyold
      i = 1
 10   moodyold = moody
      moody = 1/((1.74-2.*log10(18.7/(nre*sqrt(moodyold)) +
     +        2.*epsilon/diameter))**2)
      moody2 = 1/((1.74-2.*log10(18.7/(nre*sqrt(moodyold*1.00001)) +
     +        2.*epsilon/diameter))**2)
      err1 = moody - moodyold
      err2 = moody2 - moodyold*1.00001
      derr = (err2-err1)/(0.00001*moodyold)
      i = i + 1
      moody = moodyold - err1/derr
      moody = max(10.0**-7,moody)
      if (abs(moody-moodyold).gt.0.001.and.i.lt.10) goto 10
      if (i.gt.9) moody = .01
      return
      end
*****************************************************************
*****************************************************************
C   FLOW REGIME CALCULATION SUBROUTINE
c     subroutine determines the flow regime for Aziz Govier
c     regime is the regime
c          0 = bubble flow
c          1 = slug flow
c          2 = froth flow
c          3 = annular mist flow
c          densities in lbm/ft^3
c          rowg = gas density
c          rowair = air density
c          rowl = oil density
c          rowwater = water density
c          sigaw = air water surface tension
c          sig = oil/gas surface tension
c          x = 1 dimensionless variable
c          y = second dimensionless variable
c          vsg = gas superficial velocity ft/sec
c          vsl = oil superficial velocity ft/sec
*****************************************************************
      subroutine mregime(vsg,vsl,rowg,rowl,regime,vsg2,vsg3)
      real vsg,vsl,rowg,rowl,vsg2,vsg3
      integer regime
      real rowair,rowwater,sigaw,sig,x,y,lx,ly,l1,l2,l3
*****************************************************************
      rowair = .078
      rowwater = 62.37
      sigaw = 72.
      sig = 50.
      y = (rowl*sigaw/(rowwater*sig))**(0.25)
      x = (rowg/rowair)**(0.333333)
      x = x*y
      lx = x*vsg
      ly = y*vsl
      l1  = ((100.*ly)**.17211)/1.96
      l2  = (ly/.263)+8.6
      l3  = ((100.*ly)**(-.152))*70.
      if (lx.lt.l1) then
         regime = 0
      elseif (ly.lt.4.0) then
         regime = 2
         vsg2 = l2/x
         vsg3 = l3/x
         if (lx.lt.l2) regime = 1
         if (lx.gt.l3) regime = 3
      elseif (ly.ge.4.0) then
         if (lx.lt.26.5) regime = 1
         if (lx.ge.26.5) regime = 3
      endif
      return
      end
*****************************************************************
*****************************************************************
C  CALCULATES IN SITU LIQUID VOLUME FRACTION FOR SLUG FLOW REGIME
c     this subroutine calculates el for slug flow
c     vt = total velocity in ft/sec
c     vm = mixture velocity in ft/sec
c     vb = bubble velocity in ft/sec
c     d is pipe diameter in inches
c     rowl is density in lbm/ft^3 of oil
c     rowg is gas density in lbm/ft^3
c     vsg is apparent gas velocity in ft/sec
*****************************************************************
      subroutine elslug(el,vm,vsg,d,rowg,rowl)
      real vsg,d,rowg,rowl,el,vm
      real const,vb,vt
*****************************************************************
      const = .565
      vb = const*((d*(rowl-rowg)/rowl)**(0.5))
      vt = (1.2*vm)+vb
      el = 1.-(vsg/vt)
      return
      end
*****************************************************************
```

```
**********************************************************************
C  THIS ROUTINE CALCULATES THE IN SITU VOLUME FRACTION FOR BUBBLE REGIME
c     this subroutine calculates el for bubble flow
c     vt = total velocity in ft/sec
c     vm = mixture velocity in ft/sec
c     vb = bubble velocity in ft/sec
c     rowl is density in lbm/ft^3 of oil
c     rowg is gas density in lbm/ft^3
c     vsg is apparent gas velocity in ft/sec
c     sigma is interfacial tension
**********************************************************************
      subroutine elbubble(el,vm,vsg,sigma,rowg,rowl)
      real vsg,sigma,rowg,rowl,el,vm
      real const,vb,vt
**********************************************************************
      const = .305528
      vb = const*((sigma*(rowl-rowg)/(rowl*rowl))**(0.25))
      vt = (1.2*vm)+vb
      el = 1.-(vsg/vt)
      return
      end
**********************************************************************
**********************************************************************
C  THIS FUNCTION CALCULATES THE OIL DENSITY AT A PRESSURE AND TEMPERATURE
C  THIS TEHCNIQUE IS AN EMPIRICAL, COMPOSITIONAL METHOD FROM MCCAIN.
C  THE REOUTINE IS THE ONLY REASON FOR THE VSTAND VALUES LOOKED UP
C  IN THE DATAIN ROUTINE.
c     function figures out oil density
c     density output is lbmass/ft^3
      subroutine oildens(nc,vstand,comp,p,t,m,density)
**********************************************************************
      real vstand(nc),comp(nc),m(nc),p,t,density,diff,ma,mwap
      real numer,denom,error,olddensity,delrowp,delrowt
      real x1,x2,x3,x4,x5
      integer nc,i,j
**********************************************************************
      diff = 0.01
      ma = mwap(nc,comp,m)
      error = 0.001
      if (density.lt.0.5) density = .77
 10   olddensity = density
      if (abs(m(1)-16.043).lt.diff) then
         vstand(1) = .312+density*28.0665
         vstand(1) = vstand(1)/62.37
      endif
      if (abs(m(2)-30.07).lt.diff) then
         vstand(2) = 15.3+19.7526*density
         vstand(2) = vstand(2)/62.37
      endif
      numer = 0.0
      do i = 1,nc
         numer = numer + comp(i)*m(i)/(62.37*vstand(i))
      enddo
      density = (ma/numer)/62.37
      if (abs(density-olddensity).gt.error) goto 10
      density = density * 62.37
      delrowp = (.167+16.181*(10.**(-.0425*density)))*(p/1000.)
      delrowp = delrowp -.01*(.299+263.*(10.**(-.0603*density)))*
     + (p/1000.)**2
      density = density + delrowp
       if(density.lt.0.0) density = density-delrowp
      x1 = density**(-.951)
      x2 = t - 520.
      x2 = max(0.0,x2)
      x3 = x2**.938
      delrowt = (.00302+1.505*(x1))*(x3)
      delrowt = delrowt -(0.0216-.0233*(10.**(-.0161*density)))*
     + ((x2)**(.475))
      density = density - delrowt
      return
      end
**********************************************************************
**********************************************************************
C  THIS FUNCTION RETURNS THE API GRAVITY FOR AN FLUID OF GIVEN DENSITY
c     density must be in lbmass/cuft
**********************************************************************
      real function apigrav(density)
      real density
**********************************************************************
      apigrav = 8825.355/(density)-131.5
      return
      end
**********************************************************************
**********************************************************************
C  THIS FUNCTION RETURNS THE MOLECULAR WEIGHT OF THE GIVEN MIXTURE
**********************************************************************
      real function mwap(n,comp,m)
      real comp(n),m(n),mw
      integer n,i
**********************************************************************
      mw = 0.
      do i = 1,n
        mw = mw+comp(i)*m(i)
      enddo
      mwap = mw
      return
      end
**********************************************************************
**********************************************************************
C  FLUID VISCOSITY CORRELATIONS FROM MCCAIN
c     subroutine figures out a fluid viscosity
c     miles palke -
c     oil based dead oil of McCain
c     gas from McCain
c     nc is number of components, p is pressure(psia),t is temp (deg R)
c     comp is the composition of the fluid, and visc is the viscosity
c     m is array of molecular weights,pc and tc are critical temps and
c     pressures (rankine and psia), density is in lbmas/ft^3
c     lt liquid type (0 for gas, 1 for oil),
c     vc = critical volume of components - ft^3/lbmol
c     not yet corrected for non-hydrocarbons
**********************************************************************
      subroutine viscman(nc,p,t,comp,visc,m,tc,pc,lt,density)
      real comp(nc),visc,p,t,m(nc),tc(nc),pc(nc),visj(99),psij(99)
      real tr,viscstar,denom,numer,density,a,b,c,ma
      real densr,righthand,psi,mwap,apigrav,grav
      integer nc,i,lt
**********************************************************************
      ma = mwap(nc,comp,m)
      if (lt.eq.1) then
         grav = apigrav(density)
c        the next line just accounts for the limitations of this dead oil
c        calculation 5<api<58
         if (grav.gt.58.) grav = (5.*58.+grav)/6.
         visc =-1.+10.**(10.**(1.8653-.025086*grav-
     +       .5644*log10(t-460.)))
      endif
      if (lt.eq.0) then
         density = density/62.37
      a = (t*t*(9.379+0.01607*ma))/(209.2+19.26*ma+t)
```

```
      a = a/sqrt(t)
      b = 3.448+(986.4/t)+(ma*0.01009)
      c = 2.447-.2224*b
      visc = a*exp(b*(density**c))*1.e-4
         density = density*62.37
      endif
      return
      end
**********************************************************************
**********************************************************************
C  SUBROUTINE COMBINES TWO MASS STREAMS OF GIVEN COMPOSITONS AND MASSES
c     subroutine provides mixing of two streams based on masses to produce
c     a third stream
c     miles palke
c     stream1 and stream2 are arrays holding compositions of two streams
c          please make sure they sum to 1.
c     output is the composition of the combined stream
c     nc is number of components
c     m1 is mass flow rate of stream 1, and m2 is same for stream 2
c     mout is the mass flow rate out
**********************************************************************
      subroutine stream(nc,stream1,m1,stream2,m2,output,mout)
      real stream1(nc),stream2(nc),mout,m1,m2,output(nc)
      integer nc,i
**********************************************************************
      mout = m1+m2
      do i = 1,nc
        output(i) = (stream1(i)*m1+stream2(i)*m2)/mout
      enddo
      return
      end
**********************************************************************
**********************************************************************
C  THIS ROUTINE PERFORMS THE OUTERMOST LOOP OF THE EOS/FLASH LOOP
C  FOR A GIVEN COMPOSITION AND TEMPERATURE AND PRESSURE, THIS ROUTINE
C  CALCULATES THE PHASE RATIOS, COMPSOSITIONS, AND SPECIFIC VOLUMES
c     subroutine does the flash for a given stream
**********************************************************************
      subroutine master(l,n,z,x,y,p,t,pc,tc,omega,a,b,k,fl,
     +   fv,r,ai,bi,aij,resetk,vl2,vv2,outflcon,inflcon)
      real l,v,z(n),x(n),y(n),p,t,pc(n),tc(n),omega(n)
      real a,b,k(n),fl(n),fv(n),ferror,r,ai(n),bi(n),aij(n*n)
      real ferrorold,derrordk,kiold(99),errortol,vv2,vl2
      real x1,x2,x3,x4,outflcon,inflcon
      integer i,n,j,bob,resetk
**********************************************************************
      if (l.le.0.0.or.l.ge.1.) l = 0.5
      ferror = 1.
      vv2 = 0.
      vl2 = 0.
      errortol = outflcon
      bob = 1
      do i = 1,n
      x1 = pc(i)/p
      x2 = 1.+omega(i)
      x3 = 1.-tc(i)/t
      x4 = exp(5.37*x2*x3)
      if (resetk.or.k(i).le.errortol) k(i) = x1*x4
      enddo
10    call flasher(n,z,k,x,y,l,inflcon)
      if (l.eq.0.) then
      call rkvarinit(n,y,pc,tc,a,b,ai,bi,aij,r)
      call rkvolume(n,p,t,x1,x2,x3,r,a,b)
      vv2 = max(x1,x2,x3)
      call fugacity(n,y,fv,t,p,vv2,a,b,ai,bi,r)
      vl2 = 0.
      endif
      if (l.eq.1.) then
      call rkvarinit(n,x,pc,tc,a,b,ai,bi,aij,r)
      call rkvolume(n,p,t,x1,x2,x3,r,a,b)
      vl2 = max(x1,x2,x3)
      call fugacity(n,x,fl,t,p,vl2,a,b,ai,bi,r)
      vv2 = 0.
      endif
      if (l.gt.0.0.and.l.lt.1.0) then
      call rkvarinit(n,y,pc,tc,a,b,ai,bi,aij,r)
      call rkvolume(n,p,t,x1,x2,x3,r,a,b)
      vv2 = max(x1,x2,x3)
      call fugacity(n,y,fv,t,p,vv2,a,b,ai,bi,r)
      call rkvarinit(n,x,pc,tc,a,b,ai,bi,aij,r)
      call rkvolume(n,p,t,x1,x2,x3,r,a,b)
      if (x2.eq.0.0) x2 = 9999999.99
      if (x3.eq.0.0) x3 = 9999999.99
      vl2 = min(x1,x2,x3)
      call fugacity(n,x,fl,t,p,vl2,a,b,ai,bi,r)
      endif
      ferror = 0.0
      do i = 1,n
      ferror = ferror + abs(fl(i)-fv(i))
      enddo
      if (ferror.lt.errortol) goto 989
      if (l.ge.1..or.l.le.0.) goto 989
      do i = 1,n
       k(i) = k(i) + (k(i)*fl(i)/fv(i)-k(i))
      enddo
      do i = 1,n
      enddo
      bob = bob +1
      if (bob.lt.50) goto 10
 989   return
      end
**********************************************************************
**********************************************************************
C  THIS IS THE ACTUAL FLASH CALCULATION WITH GIVEN K RATIO VALUES
c     this subroutine performs the flash calculation
c     f is the function of l where f(l) = 0 at equilibrium
c     error is the acceptable proximity of f to 0. to end run
c     zk is the sum of z(i)*k(i) to check if l = 1.
c     zok is the sum of z(i)/k(i) to check if v = 1.
c     dfdl is the slope of f(l) - used to find next value of l by
c     newtons method (Xk+1) = Xk - F(x)/(d(F(x))/d(x))
c     i is a couniting integer
c     iters is a count which checks for lack of convergence
**********************************************************************
      subroutine flasher(n,z,k,x,y,l,inflcon)
      real f,error,z(n),k(n),x(n),y(n),l,v,zk,zok,dfdl,inflcon
      integer n,i,iters
**********************************************************************
      error = inflcon
      if (l.le.0.0.or.l.ge.1.0) l = 1.0
      l = 1.0
      iters = 1
      zk = 0.
      zok = 0.
      do i = 1,n
        zk = zk + z(i)*k(i)
        zok = zok + z(i)/k(i)
      enddo
c     if zk is less than 1, we are in liquid phase
      if (zk.lt.1.) then
```

```
         l = 1.
         do i = 1,n
            x(i)=z(i)
            y(i) = 0.
         enddo
         return
c      if zok is less than 1, we are in the vapor phase
       elseif (zok.lt.1.) then
         l = 0.
         do i = 1,n
            x(i) = 0.
            y(i) = z(i)
         enddo
         return
c      here the program really does the multistep l approximation
       else
c      program finds the f(l)
         f = 0.
  10       do i = 1,n
           f = f + (k(i)-1.)*z(i)/((k(i)+(1-k(i))*l))
c        print *, 'z(i),k(i),x(i),y(i)',z(i),k(i),x(i),y(i)
          enddo
c      if f(l) suffieciently small, we can quit
         if (abs(f).lt.error) then
           do i = 1,n
              x(i) = z(i)/(l+(1.-l)*k(i))
              y(i) = k(i)*x(i)
           enddo
           return
c      if f(l) isn't 0.0, we make a change in l
          else
c      first we find d(f)/d(l)
            do i = 1,n
               dfdl = dfdl-(k(i)-1.)*z(i)*(1.-k(i))/
     +         ((k(i)+(1.-k(i))*l)**2)
            enddo
c      now we make the new l estimate
c            write (73,*) l,f,dfdl
            l = l - f/dfdl
             if (l.lt.0.0) then
                l = 0.0
             endif
             if (l.gt.1.0) l = 1.0
c      at this point, excessive iterations stops the program
            iters = iters + 1
c            if (iters.gt.100) print *, l,f
            if (iters.gt.100) then
               print *, 'wont converge partner'
                a = a/0.
               stop
             endif
c        here, a new iteration is about to begin, so variables are initialized
            f = 0.
            dfdl = 0.
            goto 10
           endif
          endif
         end
c*********************************************************************
c*********************************************************************
C  THIS ROUTINE IS AN ANALYTIC CUBIC ROOT SOLUTION.  FROM THIELES
C  UNPUBLISHED NOTES
c*********************************************************************
      subroutine cubic(p,q,r,numroots,x1,x2,x3)
      real p,q,r,x1,x2,x3,alpha,beta,phi,pi,a,b,checker
      integer numroots
c*********************************************************************
      pi = 3.1415927
      alpha = (3.*q-(p**2.))/3.
      beta = 0.0740741*(p*p*p)-.333333333*p*q+r
      checker = ((beta*beta)/4.)+((alpha*alpha*alpha)/27.)
      if (abs(checker).lt.0.000001) then
         numroots = 2
         a = (-beta/2.)**(1./3.)
         x1 = 2.*a-(p/3.)
         x2 = -a - (p/3.)
         x3 = x2
      elseif (checker.lt.0) then
         numroots = 3
         phi = acos(-(beta/2.)*sqrt(-27./(alpha*alpha*alpha)))
         x1 = 1.154701*sqrt(-alpha)*cos(phi/3.)-(p/3.)
         x2 = 1.154701*sqrt(-alpha)*cos((phi/3.)+(.666666*pi))-(p/3.)
         x3 = 1.154701*sqrt(-alpha)*cos((phi/3.)+(1.33333*pi))-(p/3.)
      elseif (checker.gt.0) then
         numroots = 1
         a = -beta/2. +sqrt(checker)
         b = -beta/2. -sqrt(checker)
         if (a.lt.0.0) a = -((-a)**(1./3.))
         if (a.gt.0.0) a = (a**(1./3.))
         if (b.lt.0.0) b = -((-b)**(1./3.))
         if (b.gt.0.0) b = (b**(1./3.))
         x1 = -(p/3.)+a+b
         x2 = 0.
         x3 = x2
      endif
      return
      end
c*********************************************************************
c*********************************************************************
C  THIS ROUTINE INITIALIZES ALL THE REDLICH KWONG VARIABLES.
c     redlich kwong thingy - initializes the RK variables
c*********************************************************************
      subroutine rkvarinit(n,zsub,pc,tc,a,b,ai,bi,aij,r)
      real zsub(n),pc(n),tc(n),a,b,r
      real ai(n),bi(n),aij(n*n)
      integer n,i,j
c*********************************************************************
      a =0.
      b = 0.
      do i = 1, n
         ai(i) = 0.42748*r*r*tc(i)*tc(i)*tc(i)/(sqrt(tc(i))*pc(i))
         bi(i) = 0.08664*r*tc(i)/pc(i)
      enddo
      do i = 1,n
         do j = 1,n
           aij((i-1)*4.+j) = sqrt(ai(i)*ai(j))
           a = a + zsub(i)*zsub(j)*aij((i-1)*4.+j)
         enddo
         b = b +zsub(i)*bi(i)
      enddo
      do i = 1,n
      enddo
      return
      end
c*********************************************************************
c*********************************************************************
C  THIS ROUTINE FINDS THE SPECIFIC VOLUMES BASED ON THE REDLICH KWONG
C  EOS
c     redlich kwong volume finder
c*********************************************************************
```

```
      subroutine rkvolume(n,p,t,x1,x2,x3,r,a,b)
      real x1,x2,x3,a,b,p,t,pcu,rcu,qcu,r
      integer n,roots
*********************************************************************
        pcu = -r*t/p
        qcu = (1./p)*((a/sqrt(t))-b*r*t-p*b*b)
        rcu = -a*b/(p*sqrt(t))
      call cubic(pcu,qcu,rcu,roots,x1,x2,x3)
      return
      end
*********************************************************************
*********************************************************************
C  SUBROUTINES CALCULATE FUGACITIES FROM REDLICH KWONG EOS
c     subroutine calculates fugacities, and k values
*********************************************************************
      subroutine fugacity(n,zsub,f,t,p,v,a,b,ai,bi,r)
       real zsub(n),f(n),t,p,r
       real ca,cb,cbi,a,b,ai(n),bi(n)
       real v,z,x1,x2,x3,x4,x5
       integer n,i
*********************************************************************
      z = (p*v)/(r*t)
       do i = 1,n
        x1 = bi(i)*(z-1.)/b
        x2 = log(z*(1.-b/v))
        x3 = sqrt(t)/(b*r*t*t)
        x4 = (a*bi(i)/b) - 2.*sqrt(a*ai(i))
        x5 = log(1.+b/v)
        f(i)= x1-x2+x3*x4*x5
        f(i) = exp(f(i))
        f(i) = f(i)*p*zsub(i)
      enddo
      return
      end
*********************************************************************
*********************************************************************
C   ROUTINE PERFORMS SEPARATION.  PROCEDURE FULLY DESCRIBED IN MS
C   REPORT
c     this program figures out three component l/v
c     the following is the main program
*********************************************************************
      subroutine separator(n,zinput,pv,tv,m1,omega,pc,tc,m3o,
     +     m3g,salesoil,salesgas,v2,v3,outflcon,inflcon)
*********************************************************************
      real x(n),y(n),k(n),l1,l2,l3,v,pc(n)
      real tc(n),fl(n),fv(n)
      real omega(n),a,b,ai(n),bi(n),aij(n*n),gv2(n)
      real zinput(n),gv1(n),ov1(n),ov2(n),gv3(n),salesgas(n)
      real salesoil(n),zinto2(n),zinto3(n)
      real outflow,zout(n),errorcomp,errormass,dummy
      real pv(3),tv(3),m1,m1g,m1o,m2o,m2g,m3g,m3o,d1,d2
      real outflcon,inflcon
      integer i,j,n,v2,v3,resetk,bob
*********************************************************************
      r = 10.73
      bob = 1
      resetk = 1
      do i = 1,n
       k(i) = 1.0
       ov2(i) = 0.
       gv3(i) = 0.
      enddo
      m1g = 0.
      m1o = 0.
      m2o = 0.
      m3g = 0.
      m3o = 0.
      m2g = 0.
      l1 = 0.5
      l2 = 0.
      l3 = 0.
      call master(l1,n,zinput,zinto2,gv1,pv(1),tv(1),pc,tc,omega,a,b,k,
     +   fl,
     +   fv,r,ai,bi,aij,1,d1,d2,outflcon,inflcon)
      m1g = m1*(1.-l1)
      m1o = m1*l1
      do i = 1,n
      enddo
      call master(l2,n,zinto2,zinto3,gv2,pv(2),tv(2),pc,tc,
     +   omega,a,b,k,fl,fv,r,ai,bi,aij,1,d1,d2,outflcon,inflcon)
         m2o = l2*(m1o)
         m2g = m1o*(1.-l2)
      do i = 1,n
      enddo
      call master(l3,n,zinto3,salesoil,gv3,pv(3),tv(3),pc,tc,
     +   omega,a,b,k,fl,fv,r,ai,bi,aij,1,d1,d2,outflcon,inflcon)
         m3g = (1.-l3)*(m2o)
         m3o = (m2o)*l3
      do i = 1,n
      enddo
      do i = 1,n
       salesgas(i) = (m1g*gv1(i)+m2g*gv2(i)+m3g*gv3(i))/(m1g+m2g+m3g)
      enddo
       outflow = m3o + m3g + m2g + m1g
       do i = 1,n
         zout(i) = ((m1g+m2g+m3g)*salesgas(i)+m3o*salesoil(i))
     +       /(outflow)
       enddo
      m3g = m1g + m2g + m3g
      end
*********************************************************************
*********************************************************************
C  TUBING2 ROUTINE ROUTINE FIGURES OUT THE MASS FLOW RATE AND PRESSURE
C  AT WHICH GASEOUS RESERVOIR PHASE IS FIRST PRODUCED. ALSO DETERMINES
C  WHETHER STABLE INFLOW PERFORMANCE RATE IS ABOVE OR BELOW THE BREAK-
C  OUT POINT
*********************************************************************
      subroutine tubing2(n,zinput,zinj,injrate,ptop,ttop,d,pc,tc
     +     ,omega,k,fl,fv,ai,bi,aij,vstand,m,injdepth,epsilon,
     +     pcbottom,vlvpres,pwfres,pwftub,length,a,b,t2,tbottom
     +     ,zout,restemp,pbar,re,dc,thick,perm,noil,ngas,swc,sor,
     +     sgr,so,psep,tsep,mxs,outflcon,inflcon,
     +     steplength,delpmax,pwfconv,stepconv,mxmin,mxmax,chokedm,
     +     surfdiam,chokecoef,chokeconv,maxmass,pwfmax,
     +     maxpress)
*********************************************************************
      real mx2,zinput(n),zinj(n),injrate,ptop,ttop,d,pc(n),tc(n)
      real omega(n),k(n),fl(n),fv(n),ai(n),bi(n),aij(n*n),vstand(n)
      real m(n),injdepth,epsilon,pcbottom,vlvpress,pwfres,pwftub
      real length,a,b,t2,tbottom,zout(n),restemp,pbar
      real re,dc,thick,perm,noil,ngas,swc,sor,sgr,r,so,diff
      real mwap,psep(3),tsep(3),odens,outflcon
      real inflcon,rtconvg,steplength,delpmax,pwfconv
      real vl2,vv2,oilout(n),gasout(n),v2,v3
c     zinput is reservoir comp, zout is flowing comp
      integer n,itnum
      real mx,my,xs(n),ys(n),mstr,streamcom(n)
      real p2,startdep,mx22,chokeconv
      real mx2min,mx2max,delt,pwfr1,pwfr2,pwft1,pwft2,bx,by
      real mxs,stepconv,mxmin,mxmax,chokedm,surfdiam,chokecoef
```

```
      real maxmass,pwfmax,maxpress,last
      integer halt
*****************************************************************
      halt = 0
      r = 10.73
      call pwfmaxr(n,zinput,zout,pwfmax,maxmass,restemp,pc,tc,
     +     omega,a,b,k,fl,fv,r,ai,bi,aij,m,vstand,pbar,re,dc,
     +     thick,perm,noil,ngas,swc,sor,sgr,so,outflcon,
     +     inflcon,pwfconv,last)
      mxs = last
      mx2 = mxs
      call pwfmakr(n,zinput,zout,pwfres,mx2,restemp,pc,tc,
     +     omega,a,b,k,fl,fv,r,ai,bi,aij,m,vstand,pbar,re,dc,
     +     thick,perm,noil,ngas,swc,sor,sgr,so,
     +     outflcon,inflcon,pwfconv,maxmass,pwfmax)
 6    mx2 = mxs
 7    mx = mx2
      my = 0000.
      startdep = 0.0
      call stream(n,zout,mx2,zinj,injrate,streamcom,mstr)
      call choke(n,mstr,streamcom,ptop,ttop,chokedm,pc,tc
     +     ,omega,k,fl,fv,ai,bi,aij,vstand,m,a,b,
     +     psep(1),outflcon,inflcon,surfdiam,chokecoef,
     +     chokeconv,maxpress)
      call traverse(streamcom,xs,ys,mstr,my,ptop,ttop,p2,t2,d,pc,tc,
     + omega,k,fl,fv,ai,bi,aij,n,vstand,startdep,injdepth,m,epsilon,
     + mx2,outflcon,inflcon,steplength,delpmax,stepconv,
     + halt,maxpress)
      if (halt.eq.1) then
         pwftub = p2
         goto  8
      endif
      mx = mx2
      my = 0000.
      call traverse(zout,xs,ys,mx,my,p2,t2,pwftub,restemp,d,pc,
     + tc,omega,k,fl,fv,ai,bi,aij,n,vstand,injdepth,length,m,epsilon,
     + mx2,outflcon,inflcon,steplength,delpmax,stepconv,
     + halt,maxpress)
 8    mxmin = 10.
      mxmax = mxs
      if (pwftub.lt.pwfres) then
         mxmin = mxs
         mxmax = 160000.
      endif
      end
*****************************************************************
*****************************************************************
C  ROUTINE TUBING MATCHES A MASS FLOW RATE AND PRODUCED FLUID COMPOSITION
C  SO PWF FROM RESERVOIR AND TUBING ARE WITHIN A SMALL TOLERANCE OF
C  EACH OTHER
*****************************************************************
      subroutine tubing(mx2,n,zinput,zinj,injrate,ptop,ttop,d,pc,tc
     +     ,omega,k,fl,fv,ai,bi,aij,vstand,m,injdepth,epsilon,
     +     pcbottom,vlvpres,pwfres,pwftub,length,error,a,b,t2,tbottom
     +     ,zout,restemp,pbar,re,dc,thick,perm,noil,ngas,swc,sor,
     +     sgr,so,psep,tsep,outflcon,inflcon,
     +     rtconvg,steplength,delpmax,pwfconv,stepconv,mxmin,mxmax,
     +     chokedm,surfdiam,chokecoef,chokeconv,maxmass,pwfmax,
     +     maxpress)
*****************************************************************
      real mx2,zinput(n),zinj(n),injrate,ptop,ttop,d,pc(n),tc(n)
      real omega(n),k(n),fl(n),fv(n),ai(n),bi(n),aij(n*n),vstand(n)
      real m(n),injdepth,epsilon,pcbottom,vlvpress,pwfres,pwftub
      real length,error,a,b,t2,tbottom,zout(n),restemp,pbar
      real re,dc,thick,perm,noil,ngas,swc,sor,sgr,r,so,diff
      real mwap,psep(3),tsep(3),odens,outflcon
      real inflcon,rtconvg,steplength,delpmax,pwfconv
      real vl2,vv2,oilout(n),gasout(n),v2,v3
c     zinput is reservoir comp,  zout is flowing comp
      real mx,my,xs(n),ys(n),mstr,streamcom(n)
      real p2,startdep,mx22,chokedm,chokecoef,chokeconv
      real delt,pwfr1,pwfr2,pwft1,pwft2,bx,by
      real stepconv,mxmin,mxmax,surfdiam,maxmass,pwfmax,maxpress
      real last,delp1,delp2,oldmx
      integer n,itnum,halt
*****************************************************************
  1   format(3f10.0,f20.3)
      r = 10.73
  8   itnum = 0
 10   mx2 = max(mxmin,mx2)
      mx2 = min(min(mxmax,maxmass),mx2)
      oldmx = mx2
      itnum = itnum + 1
      call pwfmakr(n,zinput,zout,pwfres,mx2,restemp,pc,tc,
     +     omega,a,b,k,fl,fv,r,ai,bi,aij,m,vstand,pbar,re,dc,
     +     thick,perm,noil,ngas,swc,sor,sgr,so,outflcon,
     +     inflcon,pwfconv,maxmass,pwfmax)
      mx = mx2
      my = 0000.
      startdep = 0.0
      call stream(n,zout,mx2,zinj,injrate,streamcom,mstr)
      call choke(n,mstr,streamcom,ptop,ttop,chokedm,pc,tc
     +     ,omega,k,fl,fv,ai,bi,aij,vstand,m,a,b,
     +     psep(1),outflcon,inflcon,surfdiam,chokecoef,
     +     chokeconv,maxpress)
      if (ptop.ge.maxpress) then
         pwftub = maxpress
         goto 30
      endif
      call traverse(streamcom,xs,ys,mstr,my,ptop,ttop,p2,t2,d,pc,tc,
     + omega,k,fl,fv,ai,bi,aij,n,vstand,startdep,injdepth,m,epsilon,
     + mx2,outflcon,inflcon,steplength,delpmax,stepconv,
     + halt,maxpress)
      if (halt.eq.1) then
         mx2 = mxmin + .5*(mx2-mxmin)
         goto 10
      endif
      pcbottom = p2+vlvpres
      mx = mx2
      my = 0000.
      call traverse(zout,xs,ys,mx,my,p2,t2,pwftub,restemp,d,pc,
     + tc,omega,k,fl,fv,ai,bi,aij,n,vstand,injdepth,length,m,epsilon,
     + mx2,outflcon,inflcon,steplength,delpmax,stepconv,
     + halt,maxpress)
      if (halt.eq.1) then
         mx2 = mxmin + .5*(mx2-mxmin)
         goto 10
      endif
 30   error = (pwfres-pwftub)**2
      delp1 = pwfres-pwftub
      mx = mx2
      my = 0.0
      if (mx2.eq.maxmass.and.(pwfres.ge.pwftub)) error = -.50
      if (error.lt.rtconvg) goto 20
      mx22 = mx2 + 5.25
      call pwfmakr(n,zinput,zout,pwfres,mx22,restemp,pc,tc,
     +     omega,a,b,k,fl,fv,r,ai,bi,aij,m,vstand,pbar,re,dc,
     +     thick,perm,noil,ngas,swc,sor,sgr,so,
     +     outflcon,inflcon,pwfconv,maxmass,pwfmax)
      mx = mx22
```

```
      my = 0000.
      startdep = 0.0
      call stream(n,zout,mx22,zinj,injrate,streamcom,mstr)
      call choke(n,mstr,streamcom,ptop,ttop,chokedm,pc,tc
     +     ,omega,k,fl,fv,ai,bi,aij,vstand,m,a,b,
     +     psep(1),outflcon,inflcon,surfdiam,chokecoef,
     +     chokeconv,maxpress)
      if (ptop.ge.2.*pwfres) then
         pwftub = ptop
         goto 35
      endif
      call traverse(streamcom,xs,ys,mstr,my,ptop,ttop,p2,t2,d,pc,tc,
     + omega,k,fl,fv,ai,bi,aij,n,vstand,startdep,injdepth,m,epsilon,
     + mx22,outflcon,inflcon,steplength,delpmax,stepconv,
     + halt,maxpress)
      mx = mx22
      my = 0000.
      call traverse(zout,xs,ys,mx,my,p2,t2,pwftub,tbottom,d,pc,
     + tc,omega,k,fl,fv,ai,bi,aij,n,vstand,injdepth,length,m,epsilon,
     + mx2,outflcon,inflcon,steplength,delpmax,stepconv,
     + halt,maxpress)
      mx = mx22
      my = 0.0
 35   error2 = (pwfres-pwftub)**2
      delp2 = pwfres-pwftub
      diff = -error/((error2-error)/(mx22-mx2))
      if (pwfres-pwftub.gt.0.0.and.diff/abs(diff).lt.0.0)
     +   diff = abs(diff)
      if (pwftub.gt.pwfres.and.mxmin.lt.101.0.and.mxmax.lt.160000.)
     +   diff = -abs(diff)
      if (delp1*delp2.lt.0.0)
     +  diff = (mx22-mx2)*abs(delp1)/(abs(delp1)+abs(delp2))
      mx2 = mx2 + diff
      mx2 = max(10.,mx2)
      if (itnum.lt.7) goto 10
 20   return
      end
*****************************************************************
*****************************************************************
C  THIS ROUTINE CALCULATES KRG
*****************************************************************
      real function krgf(swc,sor,sgr,ngas,sg)
*****************************************************************
      real swc,sor,sgr,ngas,sg
*****************************************************************
      if (sg.le.sgr) krgf = 0.0
      if (sg.ge.1.-sor) krgf = 1.0
      if (sg.ge.1.-sor) return
      if (sg.gt.sgr) krgf = ((sg-sgr-swc)/(1.-swc-sgr-sor))**ngas
      return
      end
*****************************************************************
*****************************************************************
C  THIS ROUTINE CALCULATES KRO
*****************************************************************
      real function krof(swc,sor,sgr,noil,so)
*****************************************************************
      real swc,sor,sgr,noil,so
*****************************************************************
      if (so.le.sor) krof = 0.0
      if (so.ge.1.-sgr) krof = 1.0
      if (so.ge.1.-sgr) return
      if (so.gt.sor) krof = ((so-swc-sor)/(1.-sgr-swc-sor))**noil
      return
      end
*****************************************************************
*****************************************************************
C  FILE READS IN DATA - SEE THE VARIOUS INPUT FILES FOR UNITS
c     this subroutine reads in the reservoir comp file, and adjusts the z(i) is
c     there sum is not equal to 1. - save me
*****************************************************************
      subroutine datain(n,z,pc,tc,omega,ts,v2,v3,vstand,m,
     +       length,ttop,
     +       tbottom,epsilon,ctepsilon,
     +       vlvpres,dc,areal,thick,porosity,pinit,restemp,
     +       perm,swc,sor,sgr,noil,ngas,
     +       steplength,delpmax,stepconv,rtconvg,outflcon,inflcon,
     +       pwfconv,resupconv,ztol,pbarerr,
     +       chokecoef,chokeconv,maxpress)
*****************************************************************
      real z(n),zchek,pc(n),tc(n),dummy,omega(n),vstand(n),d1,d2,d3
c     zchek is the sum of the z(i)
      real m(n),length,ttop,tbottom
      real ts(3),epsilon,ctepsilon,vlvpres,dc
      real areal,perm,pinit,porosity,thick,restemp,re
      real swc,sor,sgr,Noil,Ngas
      real steplength,delpmax,stepconv,rtconvg,outflcon,inflcon
      real ztol,pwfconv,resupconv,pbarerr
      real chokecoef,chokeconv,maxpress
      integer n,si,j,v2,v3
      character*13 comname
*****************************************************************
 10   format(a13,7f13.5)
 20   format(25x,i10)
 30   format(45x,i10)
 40   format(45x,f10.3)
 50   format(45x,4f10.3)
      open (unit = 15,file = 'tubing.in',status = 'old')
      read (15,40) length,ttop,tbottom
      read (15,40) dc,epsilon,ctepsilon,vlvpres
      read (15,40) chokecoef
      open (unit = 13,file = 'comp.in',status = 'old')
      open (unit = 18,file = 'vessel.in',status = 'old')
      open (unit = 19,file = 'res.in',status = 'old')
      open (unit = 21,file = 'tol.in',status = 'old')
*****************************************************************
      read (19,40) areal,thick,porosity,pinit,restemp,perm,swc,sor,
     +    sgr,noil,ngas
      read (13,20) n
      read (13,*)
      do i = 1,n
            read (13,10) comname,z(i),pc(i),tc(i),omega(i),m(i),
     + vstand(i)
      enddo
      zchek = 0.
      do i = 1,n
         zchek = zchek + z(i)
      enddo
      if (abs(1.-zchek).gt.0.000001) then
         print *,'sums of z(i) not equal to 1.  normalizing to 1'
         do i = 1,n
           z(i) = z(i)/zchek
         enddo
      endif
      read (18,*)
      read (18,40) ts(1)
      read (18,*)
      read (18,30) v2
      read (18,40) ts(2)
      read (18,*)
```

```
      read (18,30) v3
      read (18,40) ts(3)
      read (21,40) steplength,delpmax,stepconv,rtconvg,outflcon,inflcon
      read (21,40) pwfconv,resupconv,ztol,pbarerr,chokeconv,
     + maxpress
      close (unit = 13)
      close (unit = 18)
      close (unit = 19)
      close (unit = 21)
      close (unit = 17)
      close (unit = 15)
      return
      end
*****************************************************************
*****************************************************************
C  ROUTINE RESPREP INITIALIZES RESERVOIR PARAMETERS
*****************************************************************
      subroutine resprep(areal,re,resvolume,pinit,mixdens,mpro,mbar,
     +     resmass,n,zinput,restemp,pc,tc,omega,a,b,k,fl,fv,r,
     +     ai,bi,aij,porosity,thick,m,vstand,pbar,outflcon,inflcon)
*****************************************************************
      real areal,re,resvolume,pinit,mixdens,mpro(n),mbar(n)
      real resmass,zinput(n),restemp,pc(n),tc(n),omega(n),a,b
      real fl(n),fv(n),r,ai(n),bi(n),aij(n*n),porosity,thick
      real l,oil(n),gas(n),vl2,vv2,vapdens,liqdens,m(n),k(n)
      real vstand(n),pbar,outflcon,inflcon,mwap,mres
      integer n,resetks
*****************************************************************
      areal = areal * 43560.
      re = sqrt(areal/3.141 59)
      resetks = 1
      liqdens = 50.0
      vapdens = 0.0
      mres = mwap(n,zinput,m)
      resvolume = areal*porosity*thick
      l = 0.5
      call master(l,n,zinput,oil,gas,pinit,restemp,pc,tc,omega,a,b,k,fl,
     +   fv,r,ai,bi,aij,resetks,vl2,vv2,outflcon,inflcon)
      if (vv2.gt.0.0) vapdens = 1./vv2
      if (vv2.eq.0.0) vapdens = 0.0
      call oildens(n,vstand,oil,pinit,restemp,m,liqdens)
      mixdens = liqdens*l + vapdens*(1.-l)
      do i = 1,n
        mpro(i) = 0.0
      enddo
      resmass = mixdens*resvolume
      do i = 1,n
         mbar(i) = resmass*zinput(i)/mres
      enddo
      pbar = pinit
      return
      end
*****************************************************************
*****************************************************************
C  ROUTINE PWFMAKR TAKES A MASS FLOW RATE, AND FINDS THE RESERVOIR
C  PRESSURE THAT PRODUCES THAT RATE, AND WHAT COMPOSITION IS PRODUCED
C  AT THAT RATE AND PRESSURE
*****************************************************************
      subroutine pwfmakr(n,zinput,zout,pwfres,massrate,restemp,pc,tc,
     +     omega,a,b,k,fl,fv,r,ai,bi,aij,m,vstand,pbar,re,dc,
     +     thick,perm,noil,ngas,swc,sor,sgr,so,
     +     outflcon,inflcon,pwfconv,maxmass,pwfmax)
*****************************************************************
      real zinput(n),zout(n),pc(n),tc(n),omega(n),a,b,fl(n),fv(n)
      real r,ai(n),bi(n),aij(n*n),m(n),vstand(n),k(n)
      real pwfres,massrate,restemp,pbar,vv2,vl2,liqdens,vapdens
      real mixdens,l,oil(n),gas(n),mass1,qo,qg,kro,krg,so
      real re,dc,oilvisc,gasvisc,krof,krgf,swc,sor,sgr,mwap
      real mass2,pwfres2,masserror1,masserror2,derrdpwf
      real noil,ngas,outflcon,inflcon,pwfconv
      real pwfcon2,maxmass,pwfmax,moil,mgas
      real pwfhigh,pwflow,diffhigh,difflow
      integer n,resetks,bob1
*****************************************************************
      bob1 = 0
      resetks = 1
      l = 0.5
      pwfhigh =pwfmax
      pwflow = pbar
      diffhigh = maxmass-massrate
      difflow = 0.0 - massrate
      if (massrate.gt.maxmass) then
         pwfres = pwfmax
         bob1 = -100
         goto 11
      endif
 10   pwfres = .5*(pwflow+pwfhigh)
 11   call master(l,n,zinput,oil,gas,pwfres,restemp,pc,tc,omega,a,b,k,
     +  fl,fv,r,ai,bi,aij,resetks,vl2,vv2,outflcon,inflcon)
      bob1 = bob1+1
      if (vv2.gt.0.0) vapdens = mwap(n,gas,m)/vv2
      if (vv2.eq.0.0) vapdens = 0.0
      call oildens(n,vstand,oil,pwfres,restemp,m,liqdens)
      call viscman(n,pwfres,restemp,oil,oilvisc,m,tc,pc,1,liqdens)
      if (l.lt.1.0)
     +   call viscman(n,pwfres,restemp,gas,gasvisc,m,tc,pc,0,vapdens)
      so = 1.0
      if (l.lt.1.0) so = (l/liqdens)/(l/liqdens+(1.-l)/vapdens)
      kro = krof(swc,sor,sgr,noil,so)
      krg = krgf(swc,sor,sgr,ngas,1.-so)
      moil = mwap(n,oil,m)
      qo = 0.00708*perm*kro*thick*(pbar - pwfres)
      qo = qo/(oilvisc*(log(12.*re/dc)-0.75))*liqdens*5.615/moil
      qg = 0.0
      if (l.lt.1.0) then
      mgas = mwap(n,gas,m)
      qg = 0.00708*perm*krg*thick*(pbar - pwfres)
      qg = qg/(gasvisc*(log(12.*re/dc)-0.75))*vapdens*5.615/mgas
      endif
      mass1 = qo+qg
      masserror1 = (mass1-massrate)
      if (masserror1.gt.0.0) then
         pwfhigh = pwfres
         diffhigh = masserror1
      endif
      if (masserror1.lt.0.0) then
         pwflow = pwfres
         difflow = masserror1
      endif
      masserror1 = masserror1**2
 20   call stream(n,oil,qo,gas,qg,zout,mass1)
      if (bob1.eq.-99) return
      if (masserror1.gt.pwfconv.and.bob1.lt.100) goto 10
      return
      end
*****************************************************************
*****************************************************************
C  RESUP SUBROUTINE UPDATES RESERVOIR AFTER A TIME STEPS RATE AND
C  PRODUCED COMPOSITION HAVE BEEN DETERMINED.
*****************************************************************
```

```
      subroutine resup(time,deltime,resmass,mpro,mbar,massrate,zout,
     +    zres,
     +    pbar,n,restemp,pc,tc,omega,a,b,k,fl,fv,r,ai,bi,aij,m,vstand
     +    ,resvolume,outflcon,inflcon,resupconv,pbarold)
*****************************************************************
      real time,deltime,resmass,mpro(n),mbar(n),massrate,zout(n)
      real zres(n),pbar,restemp,pc(n),tc(n),omega(n),a,b,k(n),fl(n)
      real fv(n),r,ai(n),bi(n),aij(n*n),m(n),vstand(n),resvolume
      integer n,bob2,resetks
      real pbar2,l,oil(n),gas(n),vv2,vl2,liqdens,vapdens,mixdens
      real error,error2,derr,mixdens2,mixdenstar,mwap,mres,mout
      real outflcon,inflcon,resupconv,pbarold,nnew,m1,m2
      real pbarlow,pbarhigh
*****************************************************************
      time = time + deltime
      mout = mwap(n,zout,m)
      pbarlow = 14.65
      pbarhigh = pbarold
      resmass = resmass - deltime*massrate*mout
      nnew = 0.0
      do i = 1,n
         mpro(i) = mpro(i) + deltime *massrate*zout(i)
         mbar(i) = mbar(i) - massrate*zout(i)*deltime
         nnew = nnew+mbar(i)
      enddo
      do i = 1,n
         zres(i) = mbar(i)/nnew
      enddo
      mres = mwap(n,zres,m)
      mixdenstar = resmass/resvolume
      bob2 = 0
      resetks = 1
      if (pbar.lt.pbarhigh.and.pbar.gt.pbarlow) goto 20
 10   pbar = .5*(pbarhigh+pbarlow)
 20   pbar2 = pbar + .250
      bob2 = bob2+1
      l = 1.0
      call master(l,n,zres,oil,gas,pbar,restemp,pc,tc,omega,a,b,k,fl,
     +   fv,r,ai,bi,aij,resetks,vl2,vv2,outflcon,inflcon)
      resetks = 0
      if (vv2.gt.0.0) vapdens = mwap(n,gas,m)/vv2
      if (vv2.eq.0.0) vapdens = 1.0
      call oildens(n,vstand,oil,pbar,restemp,m,liqdens)
       m1 = 0.
       m2 = 0.
       m2 = mwap(n,oil,m)
       if (l.lt.1.0) m1 = mwap(n,gas,m)
       mixdens = (liqdens*vapdens*m1/m2)-(l*liqdens*vapdens*m1/m2)
       mixdens = mixdens+(l*liqdens*vapdens)
       mixdens = mixdens/(liqdens*m1/m2-l*liqdens*m1/m2+vapdens*l)
 19   format(4f20.5)
      error = (mixdens-mixdenstar)
      if (mixdens.gt.mixdenstar) then
         pbarhigh = pbar
      endif
      if (mixdens.lt.mixdenstar) then
         pbarlow = pbar
      endif
      if (abs(error).gt.resupconv.and.bob2.lt.30) goto 10
      return
      end
*****************************************************************
*****************************************************************
C  ROUTINE OUTLOOP PERFORMS ENTIRE TIME STEPS FOR THE MODEL.  THIS
C  SUBROUTINE IS CALLED BY SUBROUTINE PROG
*****************************************************************
      subroutine outloop(pbarnew,ngasinj,injgmw,mxs,mx2,n,
     +     zreskeep,zinj,injrate,
     +     ptop,ttop,d,pc,tc,
     +     omega,k,fl,fv,ai,bi,aij,vstand,m,injdepth,epsilon,
     +     pcbottom,vlvpres,pwfres,pwftub,length,error,a,b,t2,tbottom,
     +     zout,restemp,pbarold,re,dc,thick,perm,noil,ngas,swc,sor,
     +     sgr,so,psep,tsep,outflcon,inflcon,
     +     rtconvg,steplength,delpmax,pwfconv,stepconv,mxmin,mxmax,
     +     qo,gasratesold,deltime,ztol,time,ctepsilon,pcwh,
     +     resmass,mpro,mbar,resvolume,pbarerr,chokedm,surfdiam,
     +     chokecoef,chokeconv,maxpress,resupconv)
*****************************************************************
      real ztol,zinerr,pbarnewl,mwap,resmass,mpro(n),mbar(n)
      real pbarnew,ngasinj,injgmw,mxs,mx2,resvolume,chokedm
      real zres(n),zreskeep(n),zinj(n),injrate,surfdiam
      real ptop,ttop,d,pc(n),tc(n),vl2,vv2,l,so,odens
      real omega(n),k(n),fl(n),fv(n),ai(n),bi(n),aij(n*n),vstand(n)
      real m(n),injdepth,epsilon,mstr,streamcom(n)
      real pcbottom,vlvpres,pwfres,pwftub,length,error,a,b,t2,tbottom
      real zout(n),restemp,pbarold,re,dc,thick,perm,noil,ngas,swc,sor
      real sgr,psep(3),tsep(3),outflcon,inflcon,gasratesold
      real rtconvg,steplength,delpmax,pwfconv,stepconv,mxmin,mxmax
c     zinput is reservoir comp, zout is flowing comp
      real deltime,qo,qg,r,pbarhalf,pbarnerr,oilout(n),gasout(n)
      real dce,ctepsilon,pcwh,my,xs(n),ys(n),resmassn,mpron(n),mbarn(n)
      real timen,gasratesc,pbarerr,chokecoef,ptop2,maxmass,pwfmax
      real maxpress,resupconv,lastchange,change,multiplyer
      real stoil(n),stgas(n),dummy,dummy2,moil,mgas
      integer n,bigit,littleit,v2,v3
*****************************************************************
      pbarnewl = pbarnew
      v2 =1
      v3 = 1
      t2 = (tbottom-ttop)*(injdepth)/length +ttop
      r = 10.73
      multiplyer = 0.8
      bigit = 0
 1    do i = 1,n
         zres(i) = zreskeep(i)
         mpron(i) = mpro(i)
         mbarn(i) = mbar(i)
      enddo
      bigit = bigit + 1
      littleit = 0
      timen = time
      resmassn = resmass
      pbarnew = pbarnewl + multiplyer*(pbarnew-pbarnewl)
      pbarnew = max(14.65,pbarnew)
      pbarnewl = pbarnew
      pbarhalf = 0.5*(pbarold+pbarnew)
      injgmw = mwap(n,zinj,m)
      injrate = ngasinj*(1000.*14.6)/(r*ttop)
       ptop2 = ptop
      call tubing2(n,zres,zinj,injrate,ptop2,ttop,d,pc,tc
     +     ,omega,k,fl,fv,ai,bi,aij,vstand,m,injdepth,epsilon,
     +     pcbottom,vlvpres,pwfres,pwftub,length,a,b,t2,tbottom,
     +     zout,restemp,pbarhalf,re,dc,thick,perm,noil,ngas,swc,sor,
     +     sgr,so,psep,tsep,mxs,outflcon,inflcon,
     +     steplength,delpmax,pwfconv,stepconv,mxmin,mxmax,chokedm,
     +     surfdiam,chokecoef,chokeconv,maxmass,pwfmax,
     +     maxpress)
      if (mx2.eq.0.0) mx2 = mxs
 5     littleit = littleit+ 1
      injgmw = mwap(n,zinj,m)
```

```
      injrate = ngasinj*(1000.*14.6)/(r*ttop)
*****************************************************************
      call tubing(mx2,n,zres,zinj,injrate,ptop,ttop,d,pc,tc
     +     ,omega,k,fl,fv,ai,bi,aij,vstand,m,injdepth,epsilon,
     +     pcbottom,vlvpres,pwfres,pwftub,length,error,a,b,t2,tbottom,
     +     zout,restemp,pbarhalf,re,dc,thick,perm,noil,ngas,swc,sor,
     +     sgr,so,psep,tsep,outflcon,inflcon,
     +     rtconvg,steplength,delpmax,pwfconv,stepconv,mxmin,mxmax,
     +     chokedm,surfdiam,chokecoef,chokeconv,maxmass,pwfmax,
     +     maxpress)
 21   injgmw = mwap(n,zinj,m)
      gasratesc = injrate*r*ttop/(1000.*14.6)
      call stream(n,zout,mx2,zinj,injrate,streamcom,mstr)
      call separator(n,streamcom,psep,tsep,mstr,omega,pc,tc,vl2,vv2,
     +      oilout,gasout,v2,v3,outflcon,inflcon)
      l = vl2/(vl2+vv2)
      call master(l,n,oilout,stoil,stgas,14.65,520.,
     +   pc,tc,omega,
     +   a,b,k,fl,
     +   fv,r,ai,bi,aij,1,dummy,dummy2,outflcon,inflcon)
      moil = mwap(n,stoil,m)
      call oildens(n,vstand,stoil,14.7,520.,m,odens)
      qo = l*vl2*moil/(odens*5.615)
      mgas = mwap(n,gasout,m)
      gasratesold = (vv2-injrate)*r*ttop/(1000.*14.6)
      zinerr =  0.0
      do i = 1,n
        zinerr = zinerr + sqrt((zinj(i)-gasout(i))**2)
      enddo
      if (zinerr.gt.ztol.and.littleit.lt.5) then
         do i = 1,n
            zinj(i) = gasout(i)
         enddo
         goto 5
       endif
 103  dce = sqrt(dc**2-d**2)
      my = 0.0
      if (injrate.gt.0.0) then
      call traverse2(zinj,xs,ys,injrate,my,pcbottom,t2,pcwh,ttop,
     +   dce,pc,tc,omega,k,fl,fv,ai,bi,aij,n,vstand,injdepth,
     +   0.0,m,ctepsilon,time,outflcon,inflcon,stepconv,maxpress)
      endif
      injgmw = mwap(n,zinj,m)
      call resup(timen,deltime,resmassn,mpron,mbarn,mx2,zout,
     +    zres,pbarnew,n,restemp,pc,tc,omega,a,b,k,fl,
     +    fv,r,ai,bi,aij,m,vstand
     +    ,resvolume,outflcon,inflcon,resupconv,pbarold)
      pbarnerr = (pbarnew-pbarnewl)**2
      if (pbarnerr.gt.pbarerr.and.bigit.lt.8) goto 1
      do i = 1,n
         zreskeep(i) = zres(i)
         mpro(i) = mpron(i)
         mbar(i) = mbarn(i)
      enddo
         time = timen
         resmass = resmassn
      return
      end
*****************************************************************
*****************************************************************
C  DATA ENTRY ROUTINE
c     data entry strictly for economics
*****************************************************************
      subroutine datain2(oilprice,oilesc,oilterm,gasprice,gasesc,
     + gasterm,opcost,opesc,opterm,sepcost,sepesc,septerm,horseprice,
     + horseesc,horseterm,discrate)
*****************************************************************
      real oilprice,oilesc,oilterm,gasprice,gasesc,gasterm,opcost
      real opesc,opterm,sepcost,sepesc,septerm,horseprice
      real horseesc,horseterm,discrate
*****************************************************************
 40   format(45x,f10.3)
      open (unit=15,file='econom.in',status='unknown')
      read (15,40) oilprice,oilesc,oilterm,gasprice,gasesc
      read (15,40) gasterm,opcost,opesc,opterm,sepcost,sepesc
      read (15,40) septerm,horseprice,horseesc,horseterm,discrate
      close (unit = 15)
      oilesc = oilesc/100.
      gasesc = gasesc/100.
      opesc = opesc/100.
      sepesc = sepesc/100.
      horseesc = horseesc/100.
      discrate = discrate/100.
      return
      end
*****************************************************************
*****************************************************************
C  THIS ROUTINE TAKES THE RESULTS OF A TIME STEP IN OUTLOOP TO EVALUATE
C  THE ADDITION TO NPV FROM THIS TIME STEP
c     routine evaluates economics
*****************************************************************
      subroutine econ(npw,oilcum,gascum,oilprice,oilesc,oilterm,
     + gasprice,gasesc,gasterm,opcost,opesc,opterm,
     + sepcost,sepesc,septerm,horseprice,horseesc,horseterm,
     + discrate,ngasinj,qo,qg,time,deltime,pcwh,psep,
     + peroil,pergas,perop,persep,perhorse)
*****************************************************************
      real npw,oilcum,gascum,oilprice,oilesc,oilterm
      real gasprice,gasesc,gasterm,opcost,opesc,opterm
      real sepcost,sepesc,septerm,horseprice,horseesc,horseterm
      real discrate,ngasinj,qo,qg,time,deltime,pcwh,psep
      real multiplyer,op2,gp2,oc2,sc2,hp2
      real horsepower
      real peroil,pergas,perop,persep,perhorse
*****************************************************************
      oilcum = oilcum + deltime*qo
      gascum = gascum + deltime*qg
      op2 = oilprice
      gp2 = gasprice
      oc2 = opcost
      sc2 = sepcost
      hp2 = horseprice
      if (time.lt.365.*oilterm)
     +    op2 = oilprice*(1.+oilesc*deltime/365.)
      if (time.lt.365.*gasterm)
     + gp2 = gasprice*(1.+gasesc*deltime/365.)
      if (time.lt.365.*opterm)
     + oc2 = opcost*(1.+opesc*deltime/365.)
      if (time.lt.365.*septerm)
     + sc2 = sepcost*(1.+sepesc*deltime/365.)
      if (time.lt.365.*horseterm)
     + hp2 = horseprice*(1.+horseesc*deltime/365.)
      multiplyer = (1.-discrate)**((time-.5*deltime)/365.)
      oilprice = .5*(oilprice+op2)
      gasprice = .5*(gasprice+gp2)
      opcost = .5*(opcost+oc2)
      sepcost = .5*(sepcost+sc2)
      horseprice = .5*(horseprice+hp2)
      peroil =  qo*deltime*oilprice
      pergas =  qg*deltime*gasprice
```

```
      perop = (deltime/30.4)*opcost
      horsepower = 2.23*0.0001*ngasinj*1000.*(((pcwh/psep)**0.2)-1.)
      horsepower = horsepower*deltime*24.
      perhorse = horseprice*horsepower
      persep = (qg+ngasinj)*deltime*sepcost
      npw = npw+multiplyer*(peroil+pergas-perop-perhorse-persep)
      oilprice = op2
      gasprice = gp2
      opcost = oc2
      sepcost = sc2
      horseprice = hp2
      return
      end
*****************************************************************
*****************************************************************
C  CHOKE SUBROUTINE PROVIDES AN UPSTREAM PRESSURE FOR A SPECIFIED RATE,
C  COMPOSITION, AND DOWN-STREAM PRESSURE.
c     choke model subroutine.
c     using the composition of the stream entering the separator,
c     the separator pressure, and the desired mass flow rate,
c     this subroutine predicts the upstream, well-head pressure.
c     This pressure will be used to begin the pressure traverse down the
c     tubing string.  The method is a combination of the Multi-phase
c     technique developed by Sachdeva, Schmidt, Brill and Blais and
c     the single (oleic) phase technique described on page 227 of
c     "petroleum production systems"
*****************************************************************
      subroutine choke(n,massrate,stream,ptop,ttop,chokedm,pc,tc
     +     ,omega,k,fl,fv,ai,bi,aij,vstand,m,a,b,
     +     psep,outflcon,inflcon,surfdiam,cd,chokeconv,maxpress)
*****************************************************************
      real massrate,stream(n),ptop,ttop,chokedm,pc(n),tc(n)
      real omega(n),k(n),fl(n),fv(n),ai(n),bi(n),aij(n*n),vstand(n)
      real m(n),a,b,psep,outflcon,inflcon,surfdiam,cd,chokeconv
      integer n
      real gas(n),oil(n),kratio,nexp,r,l,vl2,vv2
      real vapdens,liqdens,mixdens,mixdens2,vd1,vd2
      real junk1(n),junk2(n),ds,cpv,cvv,beta,kappa,ptemp
      real vv0,vv1,y,x,vl,vg1,vg2,yc,yfunct
      real charea,yuse,g2,massnow,mrate,ptop2,mratel
      real perror1,perror2,dptop,delptop,diamratio,lastdiff
      real nowdiff,ptopold,maxpress
      real plow,phigh,difflow,diffhigh
      real ptops(3),diffs(3)
      integer iter,index
      real qo,qg,injgmw,mwap,mflu
*****************************************************************
  10  format (f6.0,2f10.0,2f10.5)
      ptop = max(ptop,psep+.000001)
      ptop = min(ptop,4.*psep)
      ptop = psep+0.000001
      ptops(1) = ptop
      ptops(3) = maxpress
      ptopold = ptop
      iter = 0
      mflu = mwap(n,stream,m)
      diamratio = chokedm/(surfdiam*64.)
      r = 10.73
      dptop = 0.01
      yc = 0.5
      charea = 0.00000133158*((chokedm)**2)
      l = 0.5
      lastdiff = 0.0
      nowdiff = 0.0
      index = -1
20     iter = iter + 1
      do ptop = psep,maxpress,(maxpress-psep)
      index = index+2
      y = psep/ptop
      call master(l,n,stream,oil,gas,ptop,ttop,pc,tc,omega,
     +   a,b,k,fl,
     +   fv,r,ai,bi,aij,1,vl2,vv2,outflcon,inflcon)
      x = 1.-l
      if (l.gt.0.0) then
      call oildens(n,vstand,oil,ptop,ttop,m,liqdens)
      endif
      if (l.eq.1.) then
         massnow = mratel(chokedm,diamratio,ptop,psep,liqdens)
         goto 25
      endif
      call getkn(x,kratio,nexp,n,gas,ptop,ttop,pc,tc
     +     ,omega,k,fl,fv,ai,bi,aij,vstand,m,a,b,outflcon,inflcon,cpv,
     +     cvv)
      if (vv2.gt.0.0) vapdens = mwap(n,gas,m)/vv2
      if (vv2.eq.0.0) vapdens = 0.0
      vg1 = 1./vapdens
      vl = 1./liqdens
      yc = yfunct(kratio,nexp,x,vl,vg1,vg2,yc)
      massnow = mrate(y,yc,x,vg1,kratio,vl,liqdens,charea,ptop,cd)
25    perror1 = (massnow - massrate*mflu)**4
      diffs(index) = massnow-massrate*mflu
      enddo
      if (diffs(1).gt.0.0) then
         ptop = ptops(1)
         goto 30
      endif
      if (diffs(3).lt.0.0) then
         ptop = ptops(3)
         goto 30
      endif
50    iter = iter + 1
      ptop2 = .5*(ptops(1)+ptops(3))
      if (ptop2.eq.ptops(1)) goto 30
      if (ptop2.eq.ptops(3)) goto 30
51    y = psep/ptop2
      l = 0.0
      call master(l,n,stream,oil,gas,ptop2,ttop,pc,tc,omega,
     +   a,b,k,fl,
     +   fv,r,ai,bi,aij,1,vl2,vv2,outflcon,inflcon)
      if (l.gt.0.0) then
         call oildens(n,vstand,oil,ptop2,ttop,m,liqdens)
      endif
      if (l.eq.1.0) then
         massnow = mratel(chokedm,diamratio,ptop2,psep,liqdens)
         goto 26
      endif
      if (vv2.gt.0.0) vapdens = mwap(n,gas,m)/vv2
      if (vv2.eq.0.0) vapdens = 0.0
      vg1 = 1./vapdens
      vl = 1./liqdens
      x = 1.-l
      yc = yfunct(kratio,nexp,x,vl,vg1,vg2,yc)
      massnow = mrate(y,yc,x,vg1,kratio,vl,liqdens,charea,ptop2,cd)
26    perror2 = (massnow-massrate*mflu)**4
      diffs(2) = massnow - massrate*mflu
      ptops(2) = ptop2
      if (diffs(2).gt.0.0) then
         ptops(3) = ptops(2)
         diffs(3) = ptops(2)
      endif
```

```
      if (diffs(2).lt.0.0) then
         ptops(1) = ptops(2)
         diffs(1) = diffs(2)
      endif
      if (perror2.le.chokeconv) then
         ptop = ptop2
         goto 30
      endif
      if (ptops(1).eq.ptops(3)) then
        ptop = ptop2
        goto 30
      endif
      if (iter.lt.100) goto 50
  30   ptop = ptop2
      return
      end
*****************************************************************
*****************************************************************
C  ROUTINE FINDS MULTIPHASE MRATE FOR A GIVEN Y
c     This finds the mass-flow rate bashed on Sachdeva, etc.
*****************************************************************
      real function mrate(y,yc,x,vg1,kratio,vl,liqdens,charea,ptop,cd)
*****************************************************************
      real y,yc,x,vg1,kratio,vl,liqdens,charea,ptop
      real mixdens,g2,gc,cd,yuse
*****************************************************************
      gc = 32.714
      yuse = max(y,yc)
      yuse = min(yuse,1.0)
      mixdens =x*vg1*(yuse**(-1./kratio))+(1.-x)*vl
      mixdens = 1./mixdens
      g2 = vg1 - yuse*(vg1*yuse**(-1./kratio))
      g2 = ((1.-x)*(1.-yuse)/liqdens + x*kratio*(g2)/(kratio-
     +  1.))
      g2 = 2.*gc*g2*144.*ptop*mixdens*mixdens
      g2 = cd*sqrt(g2)
      mrate = g2*charea*86400.
      return
      end
*****************************************************************
*****************************************************************
C  ROUTINE FINDS Yc FOR MULTIPHASE FLOW
c     This finds the critical ratio (y) of downstream to upstream pressures
*****************************************************************
      real function yfunct(k,n,x,vl,vg1,vg2,y)
*****************************************************************
      real k,n,x,vl,vg1,vg2,yerror,yc
      real y,ynew,numer,denom
      integer ycount
*****************************************************************
      ycount = 0
      yerror = 0.00001
      ynew = y
 10   if (ycount.gt.20) y = y + (ynew-y)*.5
      if (ycount.le.20) y = ynew
      if (y.gt.1.0.or.y.lt.0.0) y = 0.5
      vg2 = vg1*(y**(-1./k))
      numer = (k/(k-1.))+(1.-x)*vl*(1.-y)/(x*vg1)
      denom = (k/(k-1.))+(n/2.)+n*(1.-x)*vl/(x*vg2)
      denom = denom +(n/2.)*(((1.-x)*vl/(x*vg2))**2.)
      ycount = ycount + 1
      ynew = numer/denom
      ynew = ynew**(k/(k-1.))
      if (abs(ynew-y).gt.yerror.and.ycount.lt.1000) goto 10
      yfunct = ynew
      return
      end
*****************************************************************
*****************************************************************
C  THIS ROUTINE CALCULATES SOME OF THE PROPERTIES THE SACHDEVA MODEL.
C  THIS IS EXTREMELY QUESTIONABLE, AND PROBABLY NEEDS TO BE REPLACED
c     this subroutine returns values of k and n for the sachdeva method.
c     this could use development, and is currently just an approximation
*****************************************************************
      subroutine getkn(x,kratio,nexp,n,gasin,pint,tint,pc,tc
     +     ,omega,k,fl,fv,ai,bi,aij,vstand,m,a,b,outflcon,inflcon,cp,
     +     cv)
*****************************************************************
      real kratio,nexp,gasin(n),pint,tint,pc(n),tc(n),omega(n)
      real k(n),fl(n),fv(n),ai(n),bi(n),aij(n*n),vstand(n)
      real m(n),a,b,outflcon,inflcon,cp,cv,cpcv,x
      integer n
      real cpstar,delt,delp,ptemp,junk1(n),junk2(n),l,vv0,vv1,vv2
      real r,vl2,cl
      integer psteps
*****************************************************************
      r = 10.73
      cpstar = 30.
      delt = 10.0
      cp = 0.0
      psteps = 10
c first we do an integral to figure out cp(p,t)
      delp = (pint - 14.65)/real(psteps)
      do i = 1,psteps
         ptemp = 14.65+(real(i)-.5)*delp
          call master(l,n,gasin,junk1,junk2,ptemp,tint-delt,
     +         pc,tc,omega,a,b,k,fl,
     +         fv,r,ai,bi,aij,1,vl2,vv0,outflcon,inflcon)
          call master(l,n,gasin,junk1,junk2,ptemp,tint,
     +         pc,tc,omega,a,b,k,fl,
     +         fv,r,ai,bi,aij,1,vl2,vv1,outflcon,inflcon)
          call master(l,n,gasin,junk1,junk2,ptemp,tint+delt,
     +         pc,tc,omega,a,b,k,fl,
     +         fv,r,ai,bi,aij,1,vl2,vv2,outflcon,inflcon)
      cp = cp + (vv2-vv1)-(vv1-vv0)
      enddo
      cp = -tint*cp*delp/(delt*delt)
      cp = cp+cpstar
      delt = 1.0
      delp = 1.0
          call master(l,n,gasin,junk1,junk2,pint,tint,
     +         pc,tc,omega,a,b,k,fl,
     +         fv,r,ai,bi,aij,1,vl2,vv0,outflcon,inflcon)
          call master(l,n,gasin,junk1,junk2,pint,tint+delt,
     +         pc,tc,omega,a,b,k,fl,
     +         fv,r,ai,bi,aij,1,vl2,vv1,outflcon,inflcon)
          call master(l,n,gasin,junk1,junk2,pint+delp,tint,
     +         pc,tc,omega,a,b,k,fl,
     +         fv,r,ai,bi,aij,1,vl2,vv2,outflcon,inflcon)
      cpcv = tint*(vv1-vv0)*(vv1-vv0)*delp/(delt*delt*(vv2-vv0))
      cv = cp + cpcv
      kratio = cp/cv
      kratio = max(kratio,1.1)
      cl = 1.15*cp
      nexp = 1.- x*cpcv/(x*cv + (1.-x)*cl)
      nexp = abs(nexp)
      return
      end
*****************************************************************
*****************************************************************
```

```
C  LIQUID PHASE CHOKE MODEL - AFTER PPS BY ECONOMIDES, ET AL.
c     this returns a mass flow rate for a given set of conditions with
c     upstream conditions being all liquid.  The constant comes from
c     petroleum production systems - but it is an approximation, and I
c     assume that the reynolds number exceeds 1E6.  This isn't to bad
c     an approximation.
**********************************************************************
      real function mratel(chokedm,diamratio,ptop,psep,liqdens)
**********************************************************************
      real chokedm,diamratio,ptop,psep,liqdens,constant
**********************************************************************
      constant = 0.995+.89891*(max(0.,(diamratio-.2))**2.63212)
      mratel = constant*31.2554*chokedm*chokedm*
     +         sqrt((ptop-psep)*liqdens)
      return
      end
**********************************************************************
**********************************************************************
C  THIS ROUTINE DETERMINES THE MAXIMUM RESERVOIR PRODUCTION, AND WHAT
C  PRESSURE THIS OCCURS AT.  IN THIS MODEL, NO PRESSURES LOWER THAN THE
C  MAX RATE PRESSURE ARE CONSIDERED.  RATES HIGHER THAN THE MAXIMUM
C  RESERVOIR RATE ARE ASSOCIATED WITH THEMAX RATE PRESSURE
**********************************************************************
      subroutine pwfmaxr(n,zinput,zout,pwfmax,maxmass,restemp,pc,tc,
     +     omega,a,b,k,fl,fv,r,ai,bi,aij,m,vstand,pbar,re,dc,
     +     thick,perm,noil,ngas,swc,sor,sgr,so,
     +     outflcon,inflcon,pwfconv,last)
**********************************************************************
      real zinput(n),zout(n),pc(n),tc(n),omega(n),a,b,fl(n),fv(n)
      real r,ai(n),bi(n),aij(n*n),m(n),vstand(n),k(n)
      real pwfres,massrate,restemp,pbar,vv2,vl2,liqdens,vapdens
      real mixdens,l,oil(n),gas(n),mass1,qo,qg,kro,krg,so
      real re,dc,oilvisc,gasvisc,krof,krgf,swc,sor,sgr,mwap
      real mass2,pwfres2,masserror1,masserror2,derrdpwf
      real noil,ngas,outflcon,inflcon,pwfconv
      real pwfcon2,pwfmax,maxmass,pwfstep,last,moil,mgas
      integer n,resetks,bob1
**********************************************************************
      bob1 = 0
      last = 10.
      resetks = 1
      pwfmax = 0.0
      maxmass = 0.0
      pwfstep = (14.65-pbar)/400.
      pwfcon2 = pwfconv
      do pwfres = pbar,14.65,pwfstep
 10   call master(l,n,zinput,oil,gas,pwfres,restemp,pc,tc,omega,a,b,k,
     +  fl,fv,r,ai,bi,aij,resetks,vl2,vv2,outflcon,inflcon)
      bob1 = bob1+1
      if (vv2.gt.0.0) vapdens = mwap(n,gas,m)/vv2
      if (vv2.eq.0.0) vapdens = 0.0
      call oildens(n,vstand,oil,pwfres,restemp,m,liqdens)
      call viscman(n,pwfres,restemp,oil,oilvisc,m,tc,pc,1,liqdens)
      if (l.lt.1.0)
     +   call viscman(n,pwfres,restemp,gas,gasvisc,m,tc,pc,0,vapdens)
      so = 1.0
      if (l.lt.1.0) so = (l/liqdens)/(l/liqdens+(1.-l)/vapdens)
      kro = krof(swc,sor,sgr,noil,so)
      krg = krgf(swc,sor,sgr,ngas,1.-so)
      moil = mwap(n,oil,m)
      qo = 0.00708*perm*kro*thick*(pbar - pwfres)
      qo = qo/(oilvisc*(log(12.*re/dc)-0.75))*liqdens*5.615/moil
      qg = 0.0
      if (l.lt.1.0) then
      mgas = mwap(n,gas,m)
      qg = 0.00708*perm*krg*thick*(pbar - pwfres)
      qg = qg/(gasvisc*(log(12.*re/dc)-0.75))*vapdens*5.615/mgas
      endif
      mass1 = qo+qg
      if (so.ge.0.9999999) last = mass1
      if (mass1.gt.maxmass) then
         maxmass = mass1
         pwfmax = pwfres
      endif
      enddo
      last = max(10.,last)
      return
      end
**********************************************************************
**********************************************************************
C  THIS ROUTINE, AND SOLVE WORK TOGETHER TO DIRECT-SOLVE MATRIX EQNS.
C  THIS ROUTINE WAS PROVIDED IN ME201A BY PROFESSOR FERZIGER.  HOWEVER,
C  IT MAY ORIGINATE FROM NUMERICAL RECIPES OR ANOTHER SOURCE.  IT IS USED
C  HERE TO CHECK FOR POLYTOPE DEGENERACY AND SOLVE NEWTON SYSTEM OF
C  EQUATIONS.
**********************************************************************
        SUBROUTINE DECOMP (NDIM,N,A,COND,IPVT,WORK,OA,Z)
**********************************************************************
C
        INTEGER NDIM,N
        DIMENSION A(NDIM,N),WORK(N),OA(NDIM,N),Z(N)
        INTEGER IPVT(N)
**********************************************************************
C
C       DECOMPOSES A REAL MATRIX BY GAUSSIAN ELIMINATION
C       AND ESTIMATES THE CONDITION OF THE MATRIX.
C
C       USE SOLVE TO COMPUTE SOLUTIONS TO LINEAR SYSTEMS.
C
C       INPUT..
C
C         NDIM = DECLARED ROW DIMENSION OF THE ARRAY CONTAINING A.
C
C         N = ORDER OF THE MATRIX.
C
C         A = MATRIX TO BE TRIANGULARIZED.
C
C       OUTPUT..
C
C         A CONTAINS AN UPPER TRIANGULAR MATRIX U AND A PERMUTED
CVERSION OF A LOWER TRIANGULAR MATRIX I-L SO THAT
C(PERMUTATION MATRIX)*A = L*U
C
C         COND = AN ESTIMATE OF THE CONDITION OF A.
C FOR THE LINEAR SYSTEM A*X =B, CHANGES IN A AND B
C MAY CAUSE CHANGES COND TIMES AS LARGE IN X.
C IF COND+1.0 .EQ. COND , A IS SINGULAR TO WORKING
C PRECISION. COND IS SET TO 1.0E+32 IF EXACT
C SINGULARITY IS DETECTED.
C
C         IPVT = THE PIVOT VECTOR.
C IPVT(K) = THE INDEX OF THE K-TH PIVOT ROW
C IPVT(N) = (-1)**(NUMBER OF INTERCHANGES)
C
C       WORK , Z .. THESE VECTORS MUST BE DECLARED AND
C       INCLUDED IN THE  CALL. THEIR INPUT CONTENTS ARE IGNORED.
C       THEIR OUTPUT CONTENTS ARE USUALLY UNIMPORTANT.
C
C         OA.. THE ORIGINAL N*N MATRIX
C
```

```
C       THE DETERMINANT OF A CAN BE OBTAINED ON OUTPUT BY
C         DET(A) = IPVT(N) * A(1,1) * A(2,2) * ... * A(N,N).
C
        REAL EK,T,ANORM,YNORM,ZNORM
        INTEGER NM1,I,J,K,KP1,KB,KM1,M
C
        DO 1 I=1,N
        DO 1 J=1,N
        OA(I,J)=A(I,J)
 1      CONTINUE
C
        IPVT(N) = 1
        IF (N.EQ.1) GO TO 80
        NM1 = N - 1
C
C       COMPUTE 1-NORM OF A
C
        ANORM = 0.0
        DO 10 J=1,N
        T=0.0
        DO 5 I=1,N
        T=T+ABS(A(I,J))
 5      CONTINUE
        IF (T.GT.ANORM) ANORM=T
 10     CONTINUE
C
C       GAUSSIAN ELIMINATION WITH PARTIAL PIVOTING
C
        DO 35 K=1,NM1
        KP1=K+1
C
C       FIND PIVOT
C
        M=K
        DO 15 I=KP1,N
        IF(ABS(A(I,K)).GT.ABS(A(M,K))) M=I
 15     CONTINUE
        IPVT(K) = M
        IF (M.NE.K) IPVT(N) =-IPVT(N)
        T = A(M,K)
        A(M,K) = A(K,K)
        A(K,K) = T
C
C       SKIP STEP IF PIVOT IS ZERO
C
        IF (T.EQ.0.0) GO TO 35
C
C       COMPUTE MULTIPLIERS
C
        DO 20 I=KP1,N
        A(I,K) = -A(I,K)/T
 20     CONTINUE
C
C       INTERCHANGE AND ELIMINATE BY COLUMNS
C
        DO 30 J=KP1,N
        T=A(M,J)
        A(M,J)=A(K,J)
        A(K,J)=T
        IF (T.EQ.0.0) GO TO 30
        DO 25 I=KP1,N
        A(I,J)=A(I,J)+A(I,K)*T
 25     CONTINUE
 30     CONTINUE
 35     CONTINUE
C
C       COND = (1-NORM OF A)*(AN ESTIMATE OF 1-NORM OF A-INVERSE)
C       ESTIMATE OBTAINED BY ONE STEP OF INVERSE ITERATION FOR THE
C       SMALL SINGULAR VECTOR. THIS INVOLVES SOLVING TWO SYSTEMS
C       OF EQUATIONS,(A-TRANSPOSE)*Y = E AND A*Z = Y WHERE E
C       IS A VECTOR OF +1 OR -1 CHOSEN TO CAUSE GROWTH IN Y.
C       ESTIMATE = (1-NORM OF Z)/(1-NORM OF Y)
C
C       SOLVE (A-TRANSPOSE)*Y = E
C
        DO 50 K=1,N
        T=0.0
        IF (K.EQ.1)GO TO 45
        KM1=K-1
        DO 40 I=1,KM1
        T=T+A(I,K)*WORK(I)
 40     CONTINUE
 45     EK=1.0
        IF(T.LT.0.0) EK=-1.0
        IF(A(K,K).EQ.0.0) GO TO 90
        WORK(K)=-(EK+T)/A(K,K)
 50     CONTINUE
        DO 60 KB=1,NM1
        K=N-KB
        T=0.0
        KP1=K+1
        DO 55 I=KP1,N
        T=T+A(I,K)*WORK(K)
 55     CONTINUE
        WORK(K)=T
        M=IPVT(K)
        IF (M.EQ.K) GO TO 60
        T=WORK(M)
        WORK(M)=WORK(K)
        WORK(K)=T
 60     CONTINUE
C
        YNORM=0.0
        DO 65 I=1,N
        YNORM=YNORM+ABS(WORK(I))
 65     CONTINUE
C
C       SOLVE A*Z = Y
C
        CALL SOLVE(NDIM,N,A,WORK,IPVT,Z)
C
        ZNORM=0.0
        DO 70 I=1,N
        ZNORM=ZNORM+ABS(Z(I))
 70     CONTINUE
C
C       ESTIMATE CONDITION
C
        COND=ANORM*ZNORM/YNORM
        IF (COND.LT.1.0) COND=1.0
        RETURN
C
C       1-BY-1
C
 80     COND=1.0
        IF (A(1,1).NE.0.0) RETURN
C
C       EXACT SINGULARITY
C
 90     COND=1.0E+32
```

```
      RETURN
      END
************************************************************************
************************************************************************
C  SEE COMMENTS FOR DECOMP.  SOURCE WAS ME201 PROF FERZIGER
************************************************************************
      SUBROUTINE SOLVE(NDIM,N,A,B,IPVT,X)
************************************************************************
C
      INTEGER NDIM,N,IPVT(N)
      DIMENSION A(NDIM,N),B(N),X(N)
C
C     SOLUTION OF LINEAR SYSTEM, A*X=B
C     DO NOT USE IF DECOMP HAS DETECTED SINGULARITY
C
C     INPUT..
C
C       NDIM = DECLARED ROW DIMENSION OF ARRAY CONTAINING A
C
C       N = ORDER OF MATRIX.
C
C       A = TRIANGULARIZED MATRIX OBTAINED FROM DECOMP
C
C       B = RIGHT HAND SIDE VECTOR
C
C       IPVT = PIVOT VECTOR OBTAINED FROM DECOMP
C
C     OUTPUT..
C
C       X = SOLUTION VECTOR,X.
C
      INTEGER KB,KM1,NM1,KP1,I,K,M
      REAL T
************************************************************************
      DO 1 I=1,N
      X(I)=B(I)
 1    CONTINUE
C
C     FORWARD ELIMINATION
C
      IF(N.EQ.1) GO TO 50
      NM1=N-1
      DO 20 K=1,NM1
      KP1=K+1
      M=IPVT(K)
      T=X(M)
      X(M)=X(K)
      X(K)=T
      DO 10 I=KP1,N
      X(I)=X(I)+A(I,K)*T
 10   CONTINUE
 20   CONTINUE
C
C     BACK SUBSTITUTION
C
      DO 40 KB=1,NM1
      KM1=N-KB
      K=KM1+1
      X(K)=X(K)/A(K,K)
      T=-X(K)
      DO 30 I=1,KM1
      X(I)=X(I)+A(I,K)*T
 30   CONTINUE
 40   CONTINUE
 50   X(1)=X(1)/A(1,1)
      RETURN
      END
************************************************************************
************************************************************************
C  THIS ROUTINE PERFORMS POLYTOPE OPTIMIZATION.  THE OPTION EXISTS
C  TO MAKE THE POLYTOPE FLIP UPHILL RATHER THAN AROUND THE CENTROID.
C  THIS SAN HELP, BUT IT MAY LEAD TO HEMSTICHING.
************************************************************************
      subroutine palketope(varcount,varindexi,varindexj,numperiods,
     + xvariables,constraints,delta,freevar)
************************************************************************
      real obfunc,xvariables(8,numperiods)
      real constraints(8,2,numperiods)
      real delta(8,numperiods)
      integer numperiods,freevar(8,numperiods)
      integer varcount,varindexi(varcount),varindexj(varcount)
      integer i,j,k,index(1+varcount),temp,count,contcount
      integer bestcount
      real xworking(8,numperiods,2+varcount)
      real xtemp(8,numperiods),xtemp2(8,numperiods)
      real centroid(8,numperiods),centroid2(8,numperiods)
      real obvalue(2+varcount),obtemp,toltot
      real alpha,beta,gamma,gamma2,bestx(8,numperiods)
      real a(varcount+1,varcount+1),b(varcount+1)
      real gradient(varcount+1),modulus,determinant
      real junka(varcount+1,varcount+1),cond
      integer ipvt(varcount+1)
      real work(varcount+1),z(varcount+1)
      real trace(varcount),theta,length
      real x2,x1,dummy2,dummy1
************************************************************************
      open (unit=32,file='centroid',status='unknown')
      open (unit=33,file='contract',status='unknown')
      alpha = 1.0
      beta = 2.0
      gamma = 0.5
      count = 0
      bestcount = 0
      do i = 1,8
         do j = 1,numperiods
            xworking(i,j,1) = xvariables(i,j)
            bestx(i,j) = 0.0
         enddo
      enddo
      do k = 1,varcount + 1
         index(k) = k
      enddo
      do k = 2,varcount+1
      do i = 1,8
         do j = 1,numperiods
            xworking(i,j,k) = xvariables(i,j)
         enddo
      enddo
      xworking(varindexi(k-1),varindexj(k-1),k) =
     + xworking(varindexi(k-1),varindexj(k-1),k) +
     + delta(varindexi(k-1),varindexj(k-1))
      enddo
  3    do k = 1,varcount+1
        do i = 1,8
         do j = 1,numperiods
          xtemp(i,j) = xworking(i,j,k)
         enddo
        enddo
        call caller(numperiods,xtemp,obfunc)
        obvalue(k) = -obfunc
```

```
      enddo
      do k = 1,varcount+1
       print *, k,
     +   (xworking(varindexi(j),varindexj(j),k),j = 1,varcount)
     +   ,obvalue(k)
      enddo
  5   count = count + 1
      do i= varcount+1,2,-1
        do j = 2,i
          if (obvalue(index(j-1)).gt.obvalue(index(j))) then
          temp = index(j)
          index(j) = index(j-1)
          index(j-1) = temp
          endif
        enddo
      enddo
      do k = 1,varcount+1
        print*, 'k,index(k)=',k,index(k)
      enddo
*****************************************************************
      do k = 1,3
      write (29,*) (xworking(varindexi(j),
     +               varindexj(j),index(k)),j = 1,varcount),
     +  obvalue(index(k))
      enddo
      write (29,*) (xworking(varindexi(j),
     +               varindexj(j),index(1)),j = 1,varcount),
     +  obvalue(index(1))
      do k = 1,varcount
       print *, 'k,varindexi,varindexj',k,varindexi(k),varindexj(k)
      enddo
      do k = 1,varcount +1
        a(k,varcount+1) = 1.0
        b(k) = obvalue(k)
        do j = 1,varcount
         a(k,j) = xworking(varindexi(j),varindexj(j),k)
        enddo
      enddo
      do k = 1,varcount+1
        do j = 1,varcount+1
          junka(k,j) = a(k,j)
        enddo
      enddo
     call decomp(varcount+1,varcount+1,junka,
     +  cond,ipvt,work,junka,z)
     call solve(varcount+1,varcount+1,junka,b,ipvt,gradient)
     print *, 'heres our plane ='
     print *, 'gradient=',gradient
     print *, 'cond = ',cond
     if (cond.gt.1000000.) then
        print *, 'no plane - have degenerated!'
        print *, 'we change the worst point'
      do i = 1,varcount
*****************************************************************
         xworking(varindexi(i),varindexj(i),index(varcount+1)) =
     +    xworking(varindexi(i),varindexj(i),index(varcount+1)) +
     +    delta(varindexi(i),varindexj(i))
      if (xworking(varindexi(i),varindexj(i),index(1))
     +   .ge.constraints(varindexi(i),2,varindexj(i))-
     +   delta(varindexi(i),varindexj(i))) then
         xworking(varindexi(i),varindexj(i),index(varcount+1))=
     +  xworking(varindexi(i),varindexj(i),index(varcount+1)) -
     +  2.*delta(varindexi(i),varindexj(i))
        endif
      enddo
      do i = 1,8
       do j = 1,numperiods
        xtemp(i,j) = xworking(i,j,index(varcount+1))
       enddo
      enddo
      call caller(numperiods,xtemp,obfunc)
     obvalue(index(varcount+1)) = - obfunc
     print *, 'back to line 5 '
     goto 5
     endif
     modulus = 0.0
     do i = 1,varcount
      modulus = modulus+gradient(i)**2
     enddo
     do i = 1,varcount
      gradient(i) = -gradient(i)/sqrt(modulus)
     enddo
     print *, 'modified gradient =',gradient
      do i = 1,8
       do j = 1,numperiods
        centroid(i,j) = 0.0
       enddo
      enddo
      do k = 1,varcount
       temp = index(k)
       do i = 1,8
        do j = 1,numperiods
         centroid(i,j) = centroid(i,j)+xworking(i,j,temp)
        enddo
       enddo
      enddo
      do i = 1,8
       do j = 1,numperiods
        centroid(i,j) = centroid(i,j)/real(varcount)
        centroid2(i,j) = centroid(i,j)
       enddo
      enddo
      print *, 'heres the centroid'
      print *,
     +   (centroid(varindexi(j),varindexj(j)),j = 1,varcount)
      write (32,*) (centroid(varindexi(j),varindexj(j)),j=1,varcount)
      print *, 'we figure out theta'
      modulus = 0.0
      do i = 1,varcount
      modulus = modulus +(centroid(varindexi(i),varindexj(i))-
     +   xworking(varindexi(i),varindexj(i),index(1)))**2
      enddo
      print *, 'modulus =',modulus
      do i = 1,varcount
       trace(i)=-(centroid(varindexi(i),varindexj(i))-
     +  xworking(varindexi(i),varindexj(i),index(1)))/sqrt(modulus)
      enddo
      print *, 'trace =',trace
      modulus = 0.0
      do i = 1,varcount
        modulus = modulus+trace(i)*gradient(i)
      enddo
      print *, 'dotproduct =',modulus
      theta = acos(modulus)
      theta = theta*180./3.141592654
      print *, 'theta =',theta
      if (abs(theta).gt.5.0) then
      length = 0.
      do i = 1,8
       do j = 1,numperiods
```

```
          centroid2(i,j) = xworking(i,j,index(varcount+1))
         enddo
        enddo
        do i = 1,varcount
          x2 = centroid(varindexi(i),varindexj(i))
          x1 = xworking(varindexi(i),varindexj(i),index(varcount+1))
          dummy2 = constraints(varindexi(i),2,varindexj(i))
          dummy1 = constraints(varindexi(i),1,varindexj(i))
             length = length + ((x2-x1)/(dummy2-dummy1))**2
       print *, length,centroid(varindexi(i),varindexj(i)),
     + xworking(varindexi(i),varindexj(i),index(varcount+1)),
     + constraints(varindexi(i),2,varindexj(i)),
     + constraints(varindexi(i),1,varindexj(i))
        print *, length,x1,x2,dummy1,dummy2
         enddo
        length = sqrt(length)
        print *, 'length = ',length
         do i =1,varcount
           centroid2(varindexi(i),varindexj(i)) =
     +      centroid2(varindexi(i),varindexj(i))  + length*
     +      gradient(i)*(constraints(varindexi(i),2,varindexj(i))-
     +       constraints(varindexi(i),1,varindexj(i)))
         enddo
C  THIS IS THE OPTION MENTIONED ABOVE.  CENTROID2 IS UPHILL, CENTROID
C  IS PROPER CENTROID.  CHANGE 4 TO 3, AND 0 TO 1, AND POLYTOPE FLOPS
C  MORE UPHILL.  CHANGE 4 TO 0 AND 0 TO 4, AND POLYTOPE IS A "STEEPEST
C  DESCENT" POLYTOPE
        do i = 1,8
         do j = 1,numperiods
           centroid2(i,j) = .25*(4.*centroid(i,j)+0.*centroid2(i,j))
         enddo
        enddo
        endif
        do i = 1,8
         do j = 1,numperiods
          xtemp(i,j) = centroid(i,j)+alpha*(centroid2(i,j)-xworking(i,j,
     +        index(varcount+1)))
          xtemp(i,j) = min(constraints(i,2,j),xtemp(i,j))
          xtemp(i,j) = max(constraints(i,1,j),xtemp(i,j))
         enddo
        enddo
        print *, 'heres the centroid2'
        print *,
     +   (centroid2(varindexi(j),varindexj(j)),j = 1,varcount)
        print *, 'heres the reflection'
        print *,
     +   (xtemp(varindexi(j),varindexj(j)),j = 1,varcount)
        call caller(numperiods,xtemp,obfunc)
         obvalue(2+varcount) = -obfunc
        bestcount = bestcount + 1
        print *, 'reflection obfunc =',-obfunc
        if (-obfunc.ge.obvalue(index(1)).and.
     +   -obfunc.le.obvalue(index(varcount))) then
         print *, 'no expand, no contract'
         obvalue(index(varcount+1)) = -obfunc
         do i = 1,8
          do j = 1,numperiods
           xworking(i,j,index(varcount+1)) = xtemp(i,j)
          enddo
         enddo
        endif
        if (-obfunc.lt.obvalue(index(1))) then
        bestcount = 0
         print *, 'reflextion good!  we expand'
         obtemp = -obfunc
          do i = 1,8
           do j = 1,numperiods
            xtemp2(i,j) = centroid(i,j)+beta*(xtemp(i,j)-centroid(i,j))
            xtemp2(i,j) = min(constraints(i,2,j),xtemp2(i,j))
            xtemp2(i,j) = max(constraints(i,1,j),xtemp2(i,j))
           enddo
          enddo
         print *, 'the expansion'
        print *,
     +   (xtemp2(varindexi(j),varindexj(j)),j = 1,varcount)
         call caller(numperiods,xtemp2,obfunc)
        print *, 'expansion obfunc =',-obfunc
        if (-obfunc.lt.obtemp) then
          print *, 'expansion succeeds'
          obvalue(index(varcount+1)) = -obfunc
          do i = 1,8
           do j = 1,numperiods
            xworking(i,j,index(varcount+1)) = xtemp2(i,j)
           enddo
          enddo
         endif
        if (-obfunc.ge.obtemp) then
          print *, 'expansion fails'
          obvalue(index(varcount+1)) = obtemp
          do i = 1,8
           do j = 1,numperiods
            xworking(i,j,index(varcount+1)) = xtemp(i,j)
           enddo
          enddo
          obfunc = -obtemp
         endif
        endif
        gamma2 = gamma
        contcount = 0
        if (-obfunc.gt.obvalue(index(varcount))) then
 10      obtemp = -obfunc
         contcount = contcount + 1
         print *, 'hey we must contract!'
         if (-obfunc.lt.obvalue(varcount+1)) then
           print *, 'contract 1'
           do i = 1,8
             do j = 1,numperiods
                xtemp2(i,j) =xworking(i,j,index(1)) + gamma2*
     +  (xtemp(i,j)-xworking(i,j,index(1)))
             xtemp2(i,j) = min(constraints(i,2,j),xtemp2(i,j))
             xtemp2(i,j) = max(constraints(i,1,j),xtemp2(i,j))
              enddo
           enddo
         endif
         if (-obfunc.ge.obvalue(varcount+1)) then
           print *, 'contract 2'
           do i = 1,8
             do j = 1,numperiods
                xtemp2(i,j) =xworking(i,j,index(1)) + gamma2*
     +  (xworking(i,j,index(varcount+1))-xworking(i,j,index(1)))
             xtemp2(i,j) = min(constraints(i,2,j),xtemp2(i,j))
             xtemp2(i,j) = max(constraints(i,1,j),xtemp2(i,j))
***********************************************************************
            enddo
           enddo
         endif
***********************************************************************
         write (33,*) (xtemp2(varindexi(j),varindexj(j)),j=1,varcount)
         call caller(numperiods,xtemp2,obfunc)
         if (-obfunc.ge.min(obtemp,obvalue(index(varcount+1)))) then
```

```
         gamma2 = gamma2 * .85
         if (contcount.gt.2) goto 20
         goto 10
        endif
        obvalue(index(varcount+1)) = -obfunc
         do i = 1,8
          do j = 1,numperiods
           xworking(i,j,index(varcount+1)) = xtemp2(i,j)
          enddo
         enddo
        endif
      if (bestcount.gt.2*varcount) goto 30
      if (count.lt.500) goto 5
      return
c     this shrinks the polytope
  20  print *, 'shrinking the polytope'
      do k = 2,varcount+1
       do i = 1,8
        do j = 1,numperiods
         xworking(i,j,index(k)) = .5*(
     + xworking(i,j,index(k))+xworking(i,j,index(1)))
          xtemp2(i,j) = xworking(i,j,index(k))
        enddo
       enddo
      call caller(numperiods,xtemp2,obfunc)
      obvalue(index(k)) = -obfunc
      enddo
      goto 5
   30 print*, 'restarting the polytope'
      toltot = 0.0
      bestcount = 0
        do i = 1,8
          do j = 1,numperiods
            toltot = toltot + abs(bestx(i,j)-xworking(i,j,index(1)))
          enddo
        enddo
        if (toltot.lt.0.1) then
           print *, 'ive converged'
           return
        endif
        do i = 1,8
          do j = 1,numperiods
             bestx(i,j) = xworking(i,j,index(1))
          enddo
        enddo
      do k = 2,varcount+1
        do i = 1,8
          do j = 1,numperiods
             xworking(i,j,index(k)) = xworking(i,j,index(1))
          enddo
        enddo
        xworking(varindexi(k-1),varindexj(k-1),index(k))=
     +  xworking(varindexi(k-1),varindexj(k-1),index(k)) +
     +  delta(varindexi(k-1),varindexj(k-1))
       if (xworking(varindexi(k-1),varindexj(k-1),index(1))
     +   .ge.constraints(varindexi(k-1),2,varindexj(k-1))-
     +   delta(varindexi(k-1),varindexj(k-1))) then
        xworking(varindexi(k-1),varindexj(k-1),index(k))=
     +  xworking(varindexi(k-1),varindexj(k-1),index(k)) -
     +  2.*delta(varindexi(k-1),varindexj(k-1))
        endif
       enddo
        do k = 2,varcount+1
         do i = 1,8
          do j = 1,numperiods
           xtemp(i,j) = xworking(i,j,index(k))
          enddo
         enddo
         call caller(numperiods,xtemp,obfunc)
         obvalue(index(k)) = -obfunc
       enddo
       print *, 'done restarting!'
       goto 5
       end
*************************************************************************
*************************************************************************
C  THIS REOUTINE PERFORMS THE GENETIC ALGORITHM OPTIMIZATION
*************************************************************************
      subroutine genetic(varcount,varindexi,varindexj,numperiods,
     + xvariables,constraints,delta,freevar,gabits,tgabits)
*************************************************************************
      real obfunc,xvariables(8,numperiods)
      real constraints(8,2,numperiods)
      real delta(8,numperiods)
      integer numperiods,freevar(8,numperiods)
      integer varcount,varindexi(varcount),varindexj(varcount)
      integer i,j,k
      integer gabits(8,numperiods),tgabits
      integer population(tgabits,tgabits)
      integer newpop(tgabits,tgabits)
      integer seed,index(varcount),list(tgabits),temp,gencount
      integer poplist1(tgabits/2),poplist2(tgabits/2)
      integer cut(tgabits/2),kold,index2(varcount)
      real xtemp(8,numperiods),dummy,fitness(tgabits),totfit
      real obvalue(tgabits),obvaluenew(tgabits)
      integer endc,temp2,lookcheck,popcompare(tgabits)
      integer indtol,lookcheck2
      integer holdworst,holdbest,maxgen,window
      integer mutcstep,crosstype
      integer cullstep,multiply
      real cullrate,cmult
      real mut1,mut2,dummy2,slope,beta,average
      integer idummy,jdummy,temp3,temp4,temp5,funccall
*************************************************************************
      print *, 'gabits',gabits
      print *, 'tgabits',tgabits
      open (unit=32,file='poptrack',status='unknown')
      open (unit=57,file='ga.in',status='old')
*************************************************************************
  4   format(50i1,f14.2)
  5   format(50i1,4f12.2)
  6   format(i3,i5,1x,6f14.2)
  7   format(45x,i10)
  8   format(45x,f10.5)
      read (57,7) holdworst,holdbest,maxgen,window
      read (57,8) mut1,mut2
      read (57,7) seed,crosstype,mutcstep
      read (57,*)
      read (57,7) cullstep
      read (57,8) cullrate
      read (57,7) multiply
      read (57,8) cmult
      funccall = 0
      gencount = 0
      do i = 1,8
       do j = 1,numperiods
        xtemp(i,j) = xvariables(i,j)
       enddo
      enddo
      do i = 1,tgabits
```

```
       list(i) = i
       do j = 1,tgabits
          dummy = ran(seed)
          k = 0
          if (dummy.gt.0.5) k = 1
          population(i,j) = k
       enddo
     enddo
     endc = tgabits
10   gencount = gencount + 1
     indtol = max(0,window-gencount)
     do i = 1,endc
       print *, 'gencount,i,indtol',gencount,i,indtol
       list(i) = i
         kold = 1
       do j = 1,varcount
         index(j) = 0
         do k = kold,kold+gabits(varindexi(j),varindexj(j))-1
           index(j) = index(j) + population(i,k)*
    +          (2**(gabits(varindexi(j),varindexj(j))-k+kold-1))
         enddo
         kold = k
         dummy = 2**gabits(varindexi(j),varindexj(j))
         dummy = dummy -1.0
         xtemp(varindexi(j),varindexj(j)) = real(index(j))/dummy
         xtemp(varindexi(j),varindexj(j)) =
    +        xtemp(varindexi(j),varindexj(j))*
    +        (constraints(varindexi(j),2,varindexj(j))-
    +        constraints(varindexi(j),1,varindexj(j)))
        xtemp(varindexi(j),varindexj(j)) =
    +        xtemp(varindexi(j),varindexj(j))+
    +        constraints(varindexi(j),1,varindexj(j))
       enddo
       lookcheck = 0
       if (gencount.gt.0) then
     open (unit=34,file='oldrun',status='unknown')
30       lookcheck = 0
         read (34,4,err=35) (popcompare(j),j=1,tgabits),
    +  (temp,j=1,50-tgabits),obfunc
         kold = 1
         do j =1,varcount
         index2(j) = 0
         do k = kold,kold+gabits(varindexi(j),varindexj(j))-1
           index2(j) = index2(j) + popcompare(k)*
    +          (2**(gabits(varindexi(j),varindexj(j))-k+kold-1))
         enddo
         kold = k
         enddo
         do j =1,varcount
        if (abs(index(j)-index2(j)).gt.indtol) lookcheck = lookcheck+1
         enddo
        if (lookcheck.gt.0) goto 30
        lookcheck = 1
       close (unit=34)
       endif
 35    if(lookcheck.eq.0)  then
     funccall = funccall + 1
     open (unit=34,file='oldrun.master',status='unknown')
 73      lookcheck2 = 0
         read (34,4,err=74) (popcompare(j),j=1,tgabits),
    +  (temp,j=1,50-tgabits),obfunc
         do j =1,tgabits
        if (abs(population(i,j)-popcompare(j)).gt.0)
    +         lookcheck2 = lookcheck2+1
         enddo
         if (lookcheck2.gt.0) goto 73
         lookcheck2 = 1
       close (unit=34)
  74   if (lookcheck2.eq.0) then
       call caller(numperiods,xtemp,obfunc)
       write (34,4) (population(i,j),j=1,tgabits),
    +  (2,j=1,50-tgabits),obfunc
        endif
       endif
       if (lookcheck.eq.0) then
     open (unit=34,file='oldrun',status='unknown')
  36   read(34,*,err=37)
       goto 36
  37     write (34,4) (population(i,j),j=1,tgabits),
    +  (2,j=1,50-tgabits),obfunc
       close (unit=34)
       endif
c       write (30,5) (population(i,j),j = 1,tgabits),
c    +     (xtemp(varindexi(k),varindexj(k)),k = 1,varcount),obfunc
     obvalue(i) = obfunc
     enddo
     do i= tgabits,2,-1
       do j = 2,i
         if (obvalue(list(j-1)).gt.obvalue(list(j))) then
         temp = list(j)
         list(j) = list(j-1)
         list(j-1) = temp
         endif
        enddo
      enddo
     temp3 = 1
     average = 0.0
     if (gencount.lt.cullstep.and.multiply.eq.1) then
       totfit = 0.0
       do i = 1,tgabits
         totfit = totfit + obvalue(i)-obvalue(list(1))
       enddo
       average = totfit
       print *, 'average,totfit =',average,totfit
      endif
     if (gencount.ge.cullstep) temp3 = tgabits*cullrate
     if (gencount.ge.cullstep) cmult = 0.0
     totfit = 0.0
     do i = 1,tgabits
        fitness(i) = 0.0
     enddo
     do i = temp3,tgabits
       totfit = totfit + obvalue(list(i))-obvalue(list(temp3))+
    + cmult*average
     enddo
     print *, 'totfit=',totfit
     if (totfit.eq.0.0) then
        fitness(list(1)) = 1./tgabits
        do i = 2,tgabits
           fitness(list(i)) = 1./tgabits + fitness(list(i-1))
        enddo
        goto 39
     endif
     fitness(list(temp3)) = 0.0
       fitness(list(temp3)) = (cmult*average+obvalue(list(temp3))-
    +   obvalue(list(temp3)))/
    +   totfit
      do i = temp3+1,tgabits
       fitness(list(i)) = (cmult*average+obvalue(list(i))-
    +   obvalue(list(temp3)))/
```

```
     +   totfit+
     +     fitness(list(i-1))
         enddo
 39     do i = 1,tgabits
          kold = 1
          do j = 1,varcount
          index(j) = 0
            do k = kold,kold+gabits(varindexi(j),varindexj(j))-1
              index(j) = index(j) + population(list(i),k)*
     +          (2**(gabits(varindexi(j),varindexj(j))-k-1+kold))
          enddo
          kold = k
          dummy = 2**gabits(varindexi(j),varindexj(j))
          dummy = dummy -1.0
          xtemp(varindexi(j),varindexj(j)) = real(index(j))/dummy
          xtemp(varindexi(j),varindexj(j)) =
     +        xtemp(varindexi(j),varindexj(j))*
     +        (constraints(varindexi(j),2,varindexj(j))-
     +        constraints(varindexi(j),1,varindexj(j)))
         xtemp(varindexi(j),varindexj(j)) =
     +        xtemp(varindexi(j),varindexj(j))+
     +        constraints(varindexi(j),1,varindexj(j))
        enddo
        if (i.eq.tgabits) then
        write (29,6) gencount,funccall,
     +     (xtemp(varindexi(k),varindexj(k)),k = 1,varcount),
     +     obvalue(list(i))
        endif
        write (30,5) (population(list(i),j),j = 1,tgabits),
     +     (2,j=1,50-tgabits),
     +     (xtemp(varindexi(k),varindexj(k)),k = 1,varcount),
     +     obvalue(list(i)),fitness(list(i))
       enddo
      do i = 1,tgabits/2
        dummy = ran(seed)
        do j = tgabits,2,-1
           if (dummy.lt.fitness(list(j))) temp = j
        enddo
        poplist1(i) = list(temp)
        dummy = ran(seed)
        do j = tgabits,2,-1
           if (dummy.lt.fitness(list(j))) temp = j
        enddo
        poplist2(i) = list(temp)
        dummy = ran(seed)
c  crossover type determined here 0 is random bit, 1 is random variable
        if (crosstype.eq.0) then
             cut(i) = dummy*tgabits
        endif
        if (crosstype.eq.1) then
             idummy = (varcount-1)*dummy+1
             idummy = max(idummy,varcount-1)
             cut(i) = 0
             do j = 1,idummy
               cut(i) = cut(i) + gabits(varindexi(j),varindexj(j))
             enddo
        endif
      enddo
      do i = 1,tgabits/2
        print *, i,poplist1(i),poplist2(i),cut(i)
      enddo
      do i = 1,tgabits/2
        do j = 1,cut(i)
         newpop(i,j) = population(poplist1(i),j)
        enddo
        do j = cut(i)+1,tgabits
         newpop(i,j) = population(poplist2(i),j)
        enddo
        do j = cut(i)+1,tgabits
         newpop(i+tgabits/2,j) = population(poplist1(i),j)
        enddo
        do j = 1,cut(i)
         newpop(i+tgabits/2,j) = population(poplist2(i),j)
        enddo
       dummy2 = ran(seed)
       print *, 'dummy2',dummy2
        if (dummy2.lt.mut1+mut2) then
           dummy = ran(seed)
           if (gencount.lt.mutcstep) then
           temp2=dummy*tgabits +1
           print *, 'tgabits,temp2',tgabits,temp2
           temp2 = max(1,temp2)
           temp = 1
           print *, 'temp,temp2',temp,temp2
           if (newpop(i,temp2).eq.1) temp = 0
           newpop(i,temp2) = temp
         print *, 'mutated'
           endif
           if (gencount.ge.mutcstep) then
           print *, 'goobered!'
           temp2 = dummy*varcount
           temp3 = temp2
           kold = 1
           if (temp2.gt.0) then
           do j = 1,temp2
             kold = kold+gabits(varindexi(j),varindexj(j))
           enddo
           endif
           temp2 = temp2+1
           index(1) = 0
           print *, 'change variable =',temp2
         do k = kold,kold+gabits(varindexi(temp2),varindexj(temp2))-1
           index(1) = index(1) + newpop(i,k)*
     +       (2**(gabits(varindexi(temp2),varindexj(temp2))-k-1+kold))
         enddo
          dummy = ran(seed)
      if (index(1).gt.1.and.index(1)+1.lt.2**(gabits(varindexi(temp2),
     +        varindexj(temp2)))) then
           temp = -1
           if (dummy.gt.0.5) temp = 1
           index(1) = index(1)+temp
        endif
        if (index(1).eq.1) index(1)= index(1)+1
        if (index(1)+1.eq.2**(gabits(varindexi(temp2),
     +       varindexj(temp2))))  index(1) = index(1) - 1
       do k = kold+gabits(varindexi(temp2),varindexj(temp2))-1,kold,-1
         temp = 2**(gabits(varindexi(temp2),varindexj(temp2))+kold-k)
         dummy = real(index(1))/real(temp)-
     +         dint(real(index(1))/real(temp))
         if (dummy.lt.0.5) newpop(i,k) = 0
         if (dummy.ge.0.5) newpop(i,k) = 1
        enddo
       endif
       endif
       if (dummy2.ge.mut1.and.dummy2.lt.mut1+mut2) then
          dummy = ran(seed)
          if (gencount.lt.mutcstep) then
          temp2=dummy*tgabits +1
          print *, 'tgabits,temp2',tgabits,temp2
          temp2 = max(1,temp2)
```

```
          temp = 1
          print *, 'temp,temp2',temp,temp2
          if (newpop(i,temp2).eq.1) temp = 0
          newpop(i,temp2) = temp
         print *, 'mutated twice'
          endif
          if (gencount.ge.mutcstep) then
 1001     temp2 = dummy*varcount
          if (temp2.eq.temp3) then
             dummy = ran(seed)
             goto 1001
          endif
          kold = 1
          if (temp2.gt.0) then
          do j = 1,temp2
            kold = kold+gabits(varindexi(j),varindexj(j))
          enddo
          endif
          temp2 = temp2+1
          index(1) = 0
          print *, 'change variable =',temp2
         do k = kold,kold+gabits(varindexi(temp2),varindexj(temp2))-1
           index(1) = index(1) + newpop(i,k)*
     +       (2**(gabits(varindexi(temp2),varindexj(temp2))-k-1+kold))
         enddo
          dummy = ran(seed)
*********************************************************************
       if (index(1).gt.1.and.index(1)+1.lt.2**(gabits(varindexi(temp2),
     +       varindexj(temp2)))) then
          temp = -1
          if (dummy.gt.0.5) temp = 1
          index(1) = index(1)+temp
        endif
        if (index(1).eq.1) index(1)= index(1)+1
        if (index(1)+1.eq.2**(gabits(varindexi(temp2),
     +      varindexj(temp2))))  index(1) = index(1) - 1
       do k = kold+gabits(varindexi(temp2),varindexj(temp2))-1,kold,-1
         temp = 2**(gabits(varindexi(temp2),varindexj(temp2))+kold-k)
         dummy = real(index(1))/real(temp)-
     +        dint(real(index(1))/real(temp))
         if (dummy.lt.0.5) newpop(i,k) = 0
         if (dummy.ge.0.5) newpop(i,k) = 1
        enddo
       endif
       endif
       dummy2 = ran(seed)
       print *, 'dummy2',dummy2
        if (dummy2.lt.mut1+mut2) then
           dummy = ran(seed)
           if (gencount.lt.mutcstep) then
           temp2=dummy*tgabits
           temp2 = max(1,temp2)
           temp = 1
           if (newpop(i+tgabits/2,temp2).eq.1) temp = 0
           newpop(i+tgabits/2,temp2) = temp
         print *, 'mutated'
           endif
           if (gencount.ge.mutcstep) then
           print *, 'goobered!'
           temp2 = dummy*varcount
           temp3 = temp2
           kold = 1
           if (temp2.gt.0) then
           do j = 1,temp2
             kold = kold+gabits(varindexi(j),varindexj(j))
           enddo
           endif
           temp2 = temp2+1
           index(1) = 0
           print *, 'change variable =',temp2
          do k = kold,kold+gabits(varindexi(temp2),varindexj(temp2))-1
            index(1) = index(1) + newpop(i,k)*
     +        (2**(gabits(varindexi(temp2),varindexj(temp2))-k-1+kold))
          enddo
           dummy = ran(seed)
        if (index(1).gt.1.and.index(1)+1.lt.2**(gabits(varindexi(temp2),
     +        varindexj(temp2)))) then
           temp = -1
           if (dummy.gt.0.5) temp = 1
           index(1) = index(1)+temp
         endif
         if (index(1).eq.1) index(1)= index(1)+1
         if (index(1)+1.eq.2**(gabits(varindexi(temp2),
     +       varindexj(temp2))))  index(1) = index(1) - 1
        do k = kold+gabits(varindexi(temp2),varindexj(temp2))-1,kold,-1
          temp = 2**(gabits(varindexi(temp2),varindexj(temp2))+kold-k)
          dummy = real(index(1))/real(temp)-
     +         dint(real(index(1))/real(temp))
          if (dummy.lt.0.5) newpop(i,k) = 0
          if (dummy.ge.0.5) newpop(i,k) = 1
         enddo
        endif
        endif
        if (dummy2.ge.mut1.and.dummy2.lt.mut1+mut2) then
           dummy = ran(seed)
           if (gencount.lt.mutcstep) then
           temp2=dummy*tgabits
           temp2 = max(1,temp2)
           temp = 1
           if (newpop(i+tgabits/2,temp2).eq.1) temp = 0
           newpop(i+tgabits/2,temp2) = temp
         print *, 'mutated twice'
           endif
           if (gencount.ge.mutcstep) then
1002       temp2 = dummy*varcount
           if (temp2.eq.temp3) then
              dummy = ran(seed)
              goto 1002
           endif
           kold = 1
           if (temp2.gt.0) then
           do j = 1,temp2
             kold = kold+gabits(varindexi(j),varindexj(j))
           enddo
           endif
           temp2 = temp2+1
           index(1) = 0
           print *, 'change variable =',temp2
          do k = kold,kold+gabits(varindexi(temp2),varindexj(temp2))-1
            index(1) = index(1) + newpop(i,k)*
     +        (2**(gabits(varindexi(temp2),varindexj(temp2))-k-1+kold))
          enddo
           dummy = ran(seed)
        if (index(1).gt.1.and.index(1)+1.lt.2**(gabits(varindexi(temp2),
     +        varindexj(temp2)))) then
           temp = -1
           if (dummy.gt.0.5) temp = 1
           index(1) = index(1)+temp
         endif
         if (index(1).eq.1) index(1)= index(1)+1
```

```
         if (index(1)+1.eq.2**(gabits(varindexi(temp2),
     +        varindexj(temp2))))  index(1) = index(1) - 1
        do k = kold+gabits(varindexi(temp2),varindexj(temp2))-1,kold,-1
          temp = 2**(gabits(varindexi(temp2),varindexj(temp2))+kold-k)
          dummy = real(index(1))/real(temp)-
     +         dint(real(index(1))/real(temp))
          if (dummy.lt.0.5) newpop(i,k) = 0
          if (dummy.ge.0.5) newpop(i,k) = 1
         enddo
        endif
        endif
 40    print *, 'i,i+tgabits/2',i,i+tgabits/2
       enddo
        print *, 'pop1,pop2,cut,new'
 50    format(2(50i1,1x),i3,1x,50i1)
       write (32,50)
       do i = 1,tgabits/2
        print *, 'i',i
        write (32,50) (population(poplist1(i),j),j=1,tgabits),
     +  (2,j=1,50-tgabits),
     +  (population(poplist2(i),j),j=1,tgabits),
     +  (2,j=1,50-tgabits),
     +  cut(i),
     +  (newpop(i,j),j=1,tgabits),
     +  (2,j=1,50-tgabits)
        write (32,50) (population(poplist2(i),j),j=1,tgabits),
     +  (2,j=1,50-tgabits),
     +  (population(poplist1(i),j),j=1,tgabits),
     +  (2,j=1,50-tgabits),
     +  cut(i),
     +  (newpop(i+tgabits/2,j),j=1,tgabits),
     +  (2,j=1,50-tgabits)
       enddo
       if (holdworst.eq.1) then
  723   format(12i1)
        temp5 = tgabits
  354   temp5 = temp5 -1
        if (obvalue(list(temp5)).eq.obvalue(list(tgabits))) goto 354
        print *, 'strings'
         write (*,723) (population(list(temp5),j),j=1,tgabits)
         write (*,723) (population(list(tgabits),j),j=1,tgabits)
c       actually extrapolating
       kold = 1
       do j = 1,varcount
        temp3 = 0
        do k = kold+gabits(varindexi(j),varindexj(j))-1,kold,-1
        temp4 = population(list(tgabits),k)+
     +       population(list(tgabits),k)
     +       -population(list(temp5),k)
        temp4 = temp4+temp3
        print *, 'k,temp4,temp3',k,temp4,temp3
        if (temp4.ge.2) temp3 = 1
        if (temp4.lt.0) temp3 = -1
        if (temp4.eq.0.or.temp4.eq.1) temp3 = 0
        newpop(tgabits/2,k) = 0
        if (temp4.eq.1.or.temp4.eq.3) newpop(tgabits/2,k) = 1
        if (temp4.eq.-1) newpop(tgabits/2,k) = 1
        enddo
        if (temp4.gt.1) then
          do k = kold,kold+gabits(varindexi(j),varindexj(j))-1
            newpop(tgabits/2,k) = 1
          enddo
        endif
        if (temp4.lt.0) then
          do k = kold,kold+gabits(varindexi(j),varindexj(j))-1
            newpop(tgabits/2,k) = 0
          enddo
        endif
        kold = kold+gabits(varindexi(j),varindexj(j))
       enddo
        print *, 'extrapolated string'
         write (*,723) (newpop(tgabits/2,j),j=1,tgabits)
       endif
************************************************************************
       if (holdbest.eq.1) then
       do j = 1,tgabits
           newpop(tgabits,j) = population(list(tgabits),j)
       enddo
       endif
       do i = 1,tgabits
         do j = 1,tgabits
           population(i,j) = newpop(i,j)
         enddo
       enddo
       if (gencount.lt.maxgen) goto 10
       return
       end
************************************************************************
```