RULE-BASED GEOLOGICAL CONSTRAINTS
FOR
TWO-DIMENSIONAL GRAVITY INVERSE MODELING

by

Steve C. S. Tsai

This thesis submitted to the Faculty and the Board of Trustees of the Colorado School of Mines in partial fulfillment of the requirements for the degree of Doctor of Philosophy, Geophysics.

Golden, Colorado
Date: 1/16/90

Signed: Steve C. S. Tsai
Steve C. S. Tsai

Approved: R. O. Hansen
Richard O. Hansen
Thesis Advisor

Golden, Colorado
Date: 1/16/90

P R Romig
Phillip R. Romig, Head
Geophysics Department

ABSTRACT

The overall goal of this research is to improve the capabilities of gravity inversion to compute general multiple-source models which are both mathematically and geologically reasonable. The technique employed is to use the rule base of an expert system shell to implement constraints in a two-dimensional gravity inversion. The resulting program contains a Marquardt inversion subroutine which performs the mathematical calculations, and a CLIPS rule-based system which incorporates constraints on geological parameters such as densities, depths, dip angles and fault throws. The rule-based system implements the constraints in the inversion calculation by interactively assigning weighting factors to the parameters. The parameterization of the model has been extensively revised to facilitate imposition of constraints. Instead of using body vertices, the program takes as parameters fault locations and dips, bed depths and dips and densities for a multiple-faulted graben and horst model. This program can invert a set of observed gravity data to obtain a best-fit and constraint-satisfied model based on a given initial model and constraints. Thus, the composite program is a

powerful two-dimensional gravity interpretation tool which can replace the conventional trial and error method.

TABLE OF CONTENTS

APPENDICES

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to thank the sponsors of the Center for Potential Fields Studies who supported this research and made my Doctoral degree financially possible. I am also grateful for the Center for Artificial Intelligence, which funded this research for the first six months.

I am forever indebted to my thesis advisor, Dr. Richard Hansen, who was irreplaceably helpful in my study at Mines, for education, inspiration and both technical and psychological support. I am also grateful to other members of my thesis committee: Dr. Norman Harthill for his geological advice and editorial comments; Drs. Ronald Knoshaug and Aaron Gordon for introducing me to the field of Artificial Intelligence; and Dr. Gregory Holden for his official review of this work.

I would also like to thank Helen Leek for help in reproducing this thesis. Finally, I must express my deep appreciation to my family for their patience and for wholly backing me all the way through school.

INTRODUCTION

Basically, there are two major methods widely used to interpret gravity anomalies quantitatively: forward modeling and inverse modeling. Forward modeling is often used to deal with complex models by trial and error. An experienced geophysicist can adjust the model by experience to minimize the difference between the model response and the observations. However, it is often hard to compromise between the geologic constraints and the geophysical data; therefore this method of forward modeling is generally very time consuming. In contrast, an unconstrained inverse modeling method is able to compute and return a mathematical model quite rapidly, but unfortunately is only capable of dealing with rather simple models in most cases.

Furthermore, the indeterminacy of interpretation of gravity data is a fact of which the geophysicists are well aware. The observed values of the field can be exactly reproduced by each of an infinite number of subsurface distributions other than the one which originally produced the anomaly (Roy, 1962). Thus, there is always uncertainty in interpreting a set of gravity data without constraints (Figure 1). The key to reducing the ambiguity in gravity interpretation is to incorporate constraints in the

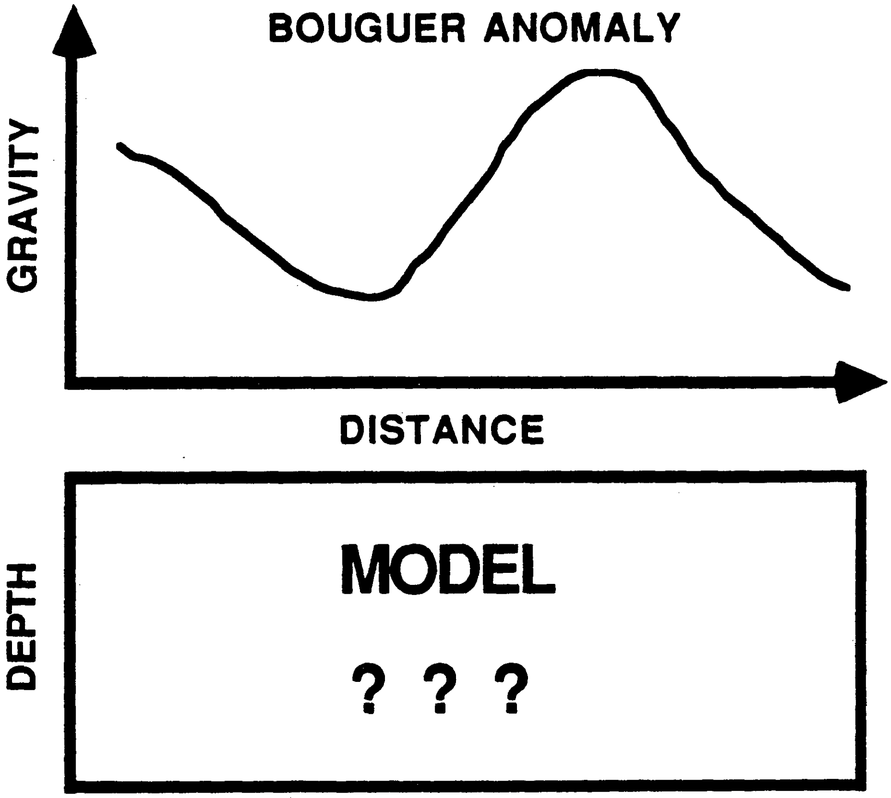


Figure 1. More than one subsurface distribution may have exactly the same gravity response.

mathematical calculation. In this research, this goal is achieved through logical control using a rule-based system which is designed to mimic the knowledge of an experienced geophysicist in inverting gravity data to obtain a gravity model.

A two-dimensional gravity inversion program was extensively modified to recognize multiple bodies and incorporate real geological constraints such as depths, dip angles, fault throws, and densities. A rule-based method was used to convert the geologic constraints into computer code. Thus, the inversion and the rule base work together: the inversion performs the computational part of the problem, and the rule base ensures that the results are geologically reasonable.

Rule-based knowledge representation centers on the use of "IF conditions THEN actions" statements. For example, when the current problem situation satisfies or matches the IF part of a rule, the action specified by the THEN part of the rule is performed. A set of rules can specify how the program should react to the changing data without requiring detailed advance knowledge about the flow of control. In a conventional program, the flow of control and use of data are predetermined by the program's code. Processing takes place in sequential steps, and branching occurs only at selected choice points (Waterman, 1986). This works well for the gravity inversion calculation, but is not appropriate for problems driven by the data, where branching is the norm rather than the exception. This is the situation in checking the results of inversion. For problems like these, rules offer the opportunity to examine the state of the calculated result at each step and react appropriately.

The rule bases need not take account of the exact method used by the inversion subroutine. Since the inversion just works as a single function callable from the rules, the rules can be solely concerned with checking to see if the result is reasonable, modifying the model if necessary and returning it to the inversion. On the other hand, the inversion routines simply perform mathematical calculations and completely ignore the logical rules. This makes the program more easily understood and more easily modified. Furthermore, the program is also more easily expanded when new geological information is available.

In this research, both the inversion and the rule-based system are essential elements. I chose the two-dimensional case as a prototype of the application of Artificial Intelligence (AI) in gravity inverse modeling. With this in mind, I sought the tools best suited to the requirements of this research.

Generally, this project is quite different from most AI applications. This project is not only rule-base oriented, but also requires a great deal of computation. Unfortunately, most AI languages or expert system shells are not designed to be strong in both building rule bases and doing complicated computations. The expert system shell named CLIPS, which was developed by NASA, and whose code is written in C, provides the capability of calling subroutines in languages other than C as its user-defined functions and has an excellent forward-chaining rule inference engine. This facility eliminated the need to rewrite the inversion program. Instead, I modified an existing Fortran inversion program, GRV2D, developed by the Center for Potential Fields Studies at the Colorado School of Mines (CPFS, 1985), to work with the CLIPS rules.

The result of this project is a program which can recognise multiple bodies and implement geological constraints in gravity modeling. The program takes parameters such as fault locations and dips, bed densities and dips and depths instead of vertices. A variety of constraints can be embedded in the program, such as the maximum range of each parameter, continuity of layer boundaries and densities, and fault throws. The program also provides the capability to keep part of the initial model unchanged or changed in a reasonable range based on information from other geophysical surveys or the user's expectations.

From this research, I conclude that the rule-based method can be used to implement geological constraints in gravity inversion. Real data testing showed that the rule base can improve the convergence of the inversion by reweighting parameters and can also ensure that a geologically reasonable result is obtained. I believe, therefore, that the program can replace the trial and error method in inverting gravity data to obtain a satisfactory gravity model.

INVERSION ALGORITHM

There are two basic components to the inversion algorithm used in this research. The first is the forward modeling algorithm, by which the gravity field associated with a given subsurface model is calculated. The other basic component is the inverse algorithm, which uses the forward model to derive the model parameters for a given set of observed data.

The forward and inverse methods chosen for this research are the two-dimensional Talwani technique (Talwani et al., 1959) and Marquardt inversion (Marquardt, 1963). These choices were motivated by the flexibility of the methods, and the fact that they have previously been used in research at the Center for Potential Fields Studies at Colorado School of Mines, so that existing software could be modified for the project. The purpose of this research was to demonstrate the applicability of rule-based systems to the inverse modeling problem, not to develop new inverse algorithms. Thus, the choice of popular, well-understood and flexible algorithms is justified.

Since both the forward and inverse methods used in this project are well-known, this section will simply review these techniques briefly.

Forward Modeling

The forward modeling algorithm used in this research is based on a variation by Corbato (1965) of work originally done by Talwani et al. (1959). The modeled bodies are approximated by n-sided polygons having infinite extent in the strike direction (Figure 2). The method is based on an equation relating volume and surface integrals. In the two-dimensional case, the surface integration reduces to a scaled line integration which can be performed using angles subtended by the various sides of the polygon at specified observation points. Convenient expressions for the calculated gravity anomaly, g , with respect to the depth of a vertex are as follows:

$$g = 2G\rho \sum_i M_i$$

where G is the gravitational constant, ρ is the density contrast, and

$$M_i = \frac{S_i}{U_i} (B_i V_i - A_i W_i)$$

$$A_i = X_{i+1} - X_i \qquad B_i = Z_{i+1} - Z_i$$

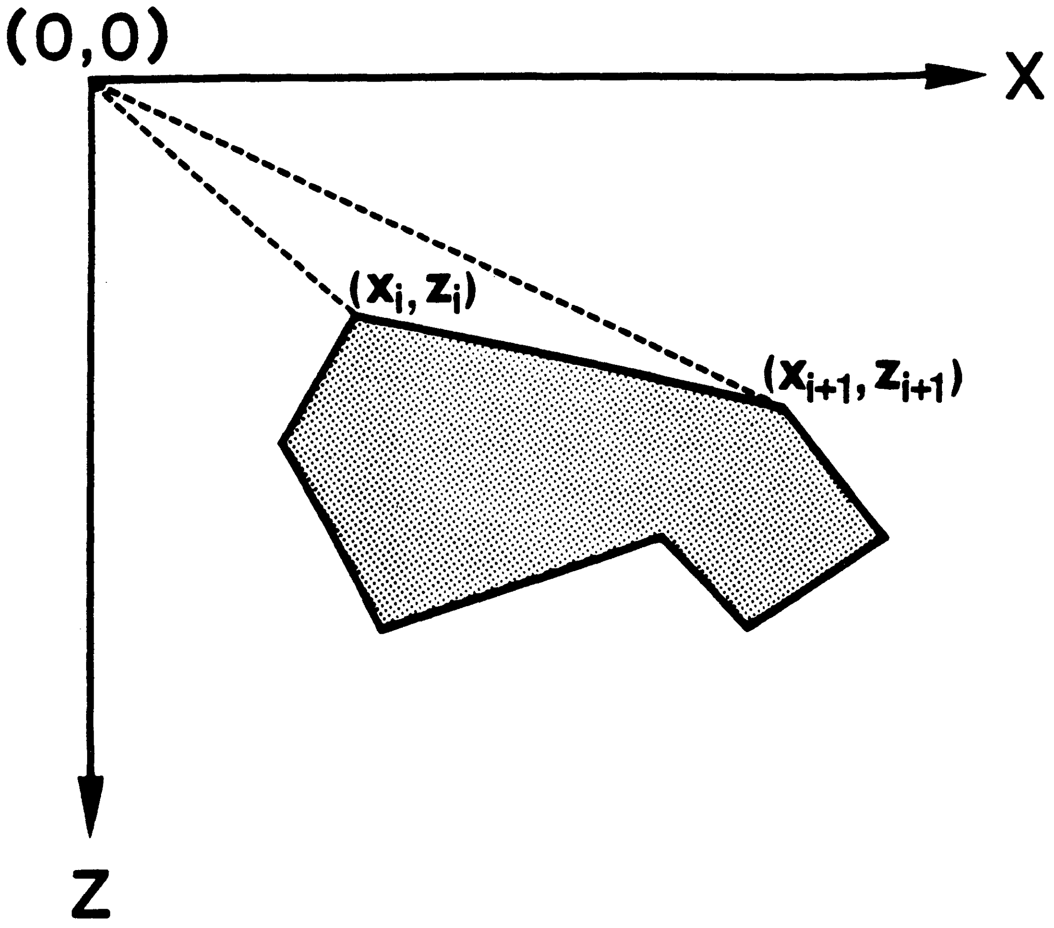


Figure 2. Notation used in the computation of the gravitational effect of a two-dimensional polygon (after Corbato, 1965).

$$R_i = X_i^2 + Z_i^2 \qquad S_i = X_i Z_{i+1} - X_{i+1} Z_i$$

$$T_i = X_i X_{i+1} + Z_i Z_{i+1} \qquad U_i = A_i^2 + B_i^2$$

$$V_i = \frac{1}{2} \ln \left(\frac{R_{i+1}}{R_i} \right) \qquad W_i = \tan^{-1} \left(\frac{S_i}{T_i} \right), \quad -\pi < W_i < \pi$$

Z is defined to be positive downward and X positive rightward. The observation points are the locations where model responses are actually calculated.

Inverse Modeling

The inversion method used in this research was developed by Marquardt (1963). Marquardt's inversion procedure essentially interpolates between the Taylor series and gradient techniques, maintaining the favorable properties of each while avoiding the pitfalls. The procedure is based on the method of Lagrange multipliers which enables the introduction of a suitable constraint on the search for minimum of the loss function, ϕ , between observed and model response values for all observation points:

$$\phi = \sum_i (O_i - M_i)^2$$

where O_i is the observed value at the ith point, and M_i is the model response at the ith point. The model response M_i can be approximately linearized by Taylor series expansion about the source body parameters $\vec{p}_0$ of m dimensions as follows:

$$M_i(\vec{p}_0 + \Delta\vec{p}) \cong M_i(\vec{p}_0) + \sum_{j=1}^{m} \left.\frac{\partial M_i}{\partial p_j}\right|_{p_0} \Delta p_j$$

where $\vec{p}_0$ is a vector of initial or current model parameters, and Δp_j is the correction to be applied. The difference between the observed values and model responses can be written as:

$$O_i = M_i(\vec{p}_0) + \sum_{j=1}^{m} \frac{\partial M_i}{\partial p_j} \Delta p_j + E_i$$

$$D_i = O_i - M_i = \sum_{j=1}^{m} \frac{\partial M_i}{\partial p_j} \Delta p_j + E_i$$

$$E_i = D_i - \sum_{j=1}^{m} \frac{\partial M_i}{\partial p_j} \Delta p_j$$

To minimize the error E_i , we define a new loss function ϕ_L:

$$\phi_L = \sum_{i=1}^{n} E_i^{2} = \sum_{i=1}^{n} \left(D_i - \sum_{j=1}^{m} \frac{\partial M_i}{\partial p_j} \Delta p_j\right)^2$$

or in matrix form

$$\Phi_l = (\vec{D} - A\,\Delta\vec{p})^2$$

Since Φ_l is a function of the corrections $\Delta\vec{p}$, a standard derivative method can be utilized to resolve for the $\Delta\vec{p}$ which will minimize Φ_l . Setting the m derivatives $\frac{\partial \Phi_l}{\partial \Delta p_j}$ = 0, we obtain an m by n system of equations which are solved simultaneously to estimate $\Delta\vec{p}$,

$$A^T A\,\Delta\vec{p} = A^T \vec{D} \qquad ; A^T \text{ is the transpose of } A$$

$$\Delta\vec{p} = (A^T A)^{-1} A^T \vec{D}$$

Marquardt modified the above equation to

$$\Delta\vec{p} = (A^T A + \lambda I)^{-1} A^T \vec{D}$$

to avoid the problem singularities in A^TA . The desired constraint is introduced via λ , sometime called the damping factor, which is added to the diagonal elements of A^TA prior to inversion. The source body parameters $\vec{p}_0$ are corrected by iteration of the varied $\Delta\vec{p}$ until the Root Mean Square Error (RMSE) of the observed values and model responses are within the required range.

A flow chart illustrating the Marquardt inversion logic is shown in Figure 3. The observed values and the initial guess model parameters are the inputs required, and must be entered prior to running the inversion. To estimate the derivative of the model response with respect to the parameter, the A matrix is calculated for the Talwani forward model response of the current parameters $\vec{p}$ and for the perturbed parameters $\vec{p} + d\vec{p}$. The difference between these responses is then divided by $d\vec{p}$. The vector $d\vec{p}$ contains the small increment for each parameter.

The choice of an appropriate damping factor λ is arguably the heart of the Marquardt technique. Conventionally, the trace of A^TA is chosen as the initial λ and then alternatively decreased or increased by a factor to ensure that the calculated model has the smallest response-residual compared with the observations in the neighborhood of the current value. A more detailed discussion of the choice of damping factor to optimize Marquardt inversion is described in the papers by Hoerl and Kennard (1970), Inman (1975), and Marquardt (1970).

The Marquardt equation shown above is used to calculate the correction term $\Delta\vec{p}$. First, the inverse matrix is calculated using Guassian elimination. Then the Talwani

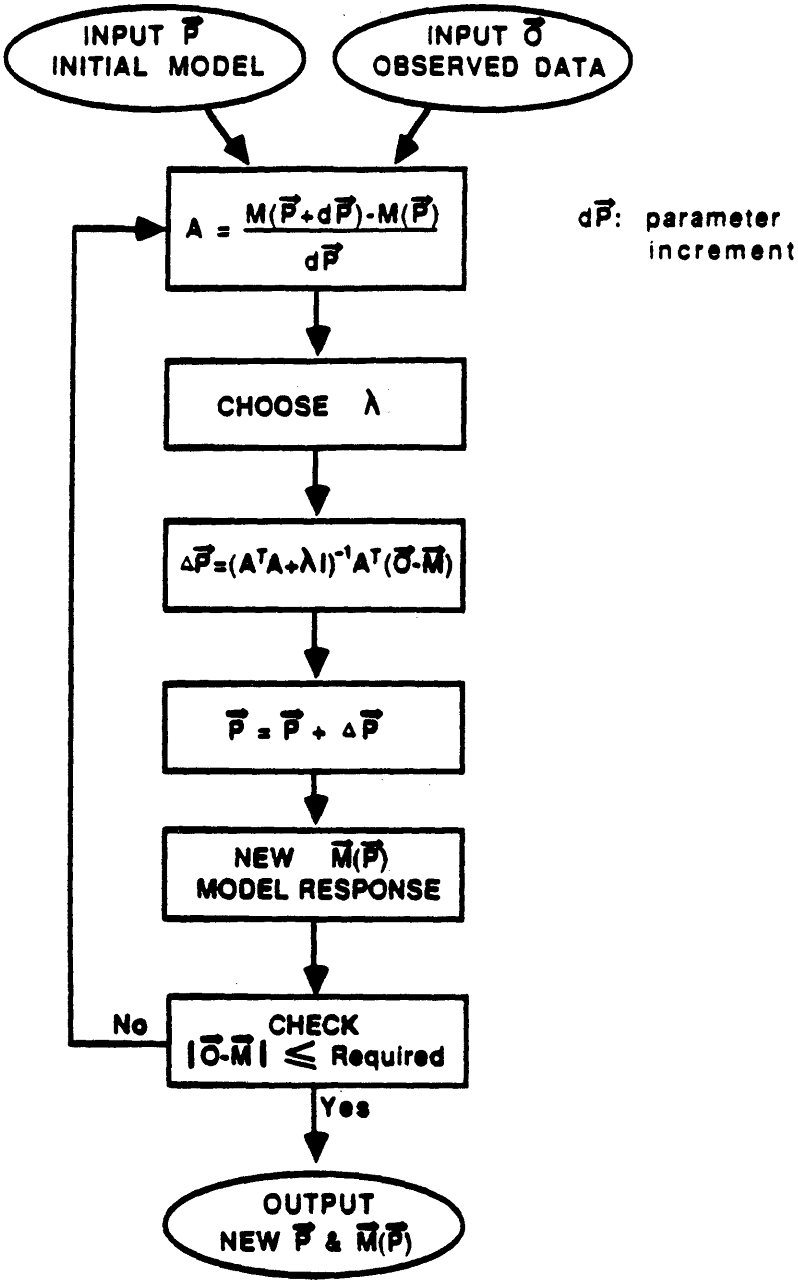


Figure 3. Flow chart of Marquardt inversion.

forward model routine is called again to compute the response of the updated model. There are two ways to determine whether the inversion routines should proceed to further iteration or return the updated model to the caller: one is based on the fit of the current model; i.e., if the fit is within the required range, then the model is returned. The other is based on the number of iterations, i.e., if the number of iterations is beyond a preset limit then the current model is returned. The latter condition is used to stop the inversion calculation from looping endlessly.

Generally, the solution of the Marquardt inversion equation improves the fit at each iteration, especially during the first few iterations. However, the inversion routine can not make much improvement by further iteration if the final calculated model is unsatisfactory while using the same given initial model. The method by which the inversion updates the model rather confines the final model to the neighborhood of the initial model; thus, the best choice of an initial model is important to find a reasonable model using the inversion technique. This problem can be resolved with the help of a rule base, and that is the topic discussed in the following chapter.

RULE-BASED SYSTEM

Introduction

In recent years, Artificial Intelligence researchers have achieved many significant successes in developing powerful computer programs known as "rule-based systems" or "knowledge-based systems" or "expert systems". These programs are designed to apply human expertise to solve specific problems and can replace human experts at a high level. They are quite different from conventional computer programs.

Generally, most rule-based systems differ from conventional computer programs in that they use symbolic representation, symbolic inference, and heuristic search (Hayes-Roth et al., 1983). One basic difference is that rule-based systems manipulate symbolic data while conventional programs manipulate numeric data. Rule-based systems represent knowledge symbolically, as sets of symbols that stand for problem concepts. To solve a problem, a rule-based system manipulates these symbols rather than performing standard mathematical computations. This is not to say that rule-based systems don't do mathematics; rather the emphasis is on manipulating symbols. Most importantly, rule-based systems apply domain-specific knowledge to guide the solution search process instead of blindly searching.

The knowledge in a rule-based system is organized in a way that separates the knowledge about the problem domain from the system's other knowledge, such as general knowledge about how to solve the problem or knowledge about how to interact with the user. The collection of domain knowledge is called the rule base or knowledge base, while the general problem-solving knowledge is called the inference engine (Waterman, 1986).

In a rule-based system, the domain knowledge is represented as sets of rules, e.g., "IF conditions THEN actions" statements, that are checked against a collection of facts or knowledge about the current situation of the problem. A rule interpreter compares the predicates of rules, i.e., the IF portions of rules, with the facts stored in the data base, and executes the rules whose predicates match the facts. A rule's execution may change the status of facts in the data base by adding a new fact, and the new facts can be used to form matches with the predicates of other rules.

Basically, there are two rule inference methods that can be used in a rule-based system; one is called forward chaining, and the other backward chaining. The forward-chaining method proceeds in the direction of IF => THEN. It works from the current state to the goal state, i.e., a rule is executed whenever its predicates are matched by the current facts in the data base. It proceeds initially from matching predicates to making conclusions from the rules. In contrast, the backward-chaining method proceeds in the reverse direction of IF => THEN. It works from the goal state to the current state, i.e., it assumes the goals of a rule first and then tries to determine if its predicates are matched by the current facts. If the predicates are true then the rule is executed; otherwise, the inference engine will try to achieve the goals of another rule. These two inference algorithms can be applied separately or mixed together in a single rule-based system, depending on the tool used and the builder's choice.

The rule-based system in this research is quite different from most rule-based system applications. It is required to deal with a great deal of arithmetic computation, and to make decisions based on both calculations and heuristics of gravity modeling expertise. Applying a rule-based system in this project is expected to reduce the task of programming. The rule base in this project is intended to help the inversion to find a solution compatible with the user's expectation, and to check and modify the model returned from the inversion. Thus, human expertise is used to correct the deficiencies of general inversion techniques. This part of the gravity modeling task is more rule-oriented than mathematics-oriented. Thus, rule-based systems are much more appropriate to implement this expertise into computer code than FORTRAN. This is not say that FORTRAN could not be used, but it might result in ten times or even a hundred times more complexity than a rule-based system. Furthermore, writing rule-like programs in a rule-based system is much more effective and efficient than a FORTRAN program, and is better suited to further rule base expansion if needed.

Expert System Shells

To construct a rule-based system efficiently and effectively, the choice of a correct expert system tool is always crucial. Generally, expert system shells provide special facilities for building rule-based systems rather easily and fast, but are often less flexible than programming languages such as LISP or PROLOG.

An expert system shell is a tool for developing rule-based systems. Basically, it is a program designed to act as a rule parser or rule interpreter, and offers a rule inference engine and a data base for storing the facts. Different expert system shells have a different syntaxes for writing the rules and different facilities which can be applied. Most expert system shells use a rule representation paradigm which represents the domain-knowledge as an English-like IF/THEN statements. These rules can be written either using a screen editor or a word processor embedded in the expert system shell.

Special Requirements for This Project

This project is not only rule-base oriented, but also requires a great deal of computation. In this project, both the inversion and the rule-based system are essential elements, each taking responsibility for implementing part of the task. The inversion program used in this project is written in FORTRAN, and would be redundant to rewrite in a language such as LISP. Thus, an expert system shell which can work with other languages, especially FORTRAN, is a basic requirement for this project.

The above requirement is not a difficulty for most commercial expert system shells since it can be simply satisfied through the use of operating system calls. But this mechanism proved to be inefficient, and it was difficult to make the programs interact conveniently. Therefore, an expert system shell whose source code can be compiled and linked with FORTRAN code is required for this project.

CLIPS, an expert system shell developed by NASA, satisfies both requirements for this project, and so it was selected.

Features of CLIPS

CLIPS, whose source code is written in C, provides an excellent forward-chaining rule inference engine and allows us to call subroutines in other languages than C as its user-defined functions. The basic elements of CLIPS are (COSMIC, 1987):

1. fact-list: global memory for data,
2. production memory: contains all the rules or productions,
3. inference engine: controls overall production execution.

The fact-list will be also called "data base" in this thesis. A program written in CLIPS consists of rules and facts. Facts are required to drive the execution of rules; the inference engine decides which rules should be executed. A CLIPS rule is analogous to an IF/THEN statement in a language such as FORTRAN. An IF/THEN rule can be expressed in natural language as follows:

```
IF certain conditions are true
THEN execute the following actions
```

We can either type rules directly into CLIPS or load rules in from a file of rules created by a word processor. For example, the pseudocode for a rule is

```
IF there is a duck
THEN make a quack
```

The rule expressed in CLIPS syntax is

```
(defrule duck-quack
   (duck)
=>
   (assert (quack)))
```

A CLIPS rule consists of the following essential parts:

1. The entire rule must be surrounded by parentheses.
2. The rule must start with a "defrule" keyword.
3. Following defrule must be the name of the rule. The name can be any valid symbolic atom, e.g., duck-quack in this example.
4. After the name of the rule are one or more conditional elements.

Each conditional element, which will be called a "predicate" hereafter, consists of one or more fields. In the duck-quack rule, the predicate is (duck) and the only "field" is duck.

CLIPS attempts to match predicates of rules against facts in the data base. If all the predicates of a rule match facts, the rule is triggered.

5. The symbol => that follows the predicate in a rule is called an arrow. The arrow is a symbol representing the THEN part of an IF/THEN rule.
6. The last part of a rule is the list of actions that will be executed when the rule fires. In this example, the one action is to assert the fact

(quack). (COSMIC, 1987).

The rule base in this project calls inversion subroutines, checks the calculated results and modifies the models if needed. The forward-chaining algorithm in CLIPS is sufficient to satisfy the requirements of this project.

CLIPS is developed in C. It is easily combined with user functions written in C, and, the most significant characteristic, can be integrated with other languages. For example, FORTRAN can be used for user-defined functions called from CLIPS exactly as the built-in function "assert" in the above rule example, and CLIPS may even be embedded within a FORTRAN main program. Furthermore, those C and FORTRAN subroutines can be compiled and linked with the CLIPS source code. In this project, I include FORTRAN subroutines as user-defined functions in CLIPS. The FORTRAN subroutines are called from the CLIPS rules like any other user-defined functions in CLIPS, and arguments can be passed in both directions, i.e., from CLIPS to FORTRAN or from FORTRAN to CLIPS.

Another significant characteristic of CLIPS is that a rule can be retriggered for any number of times. This capability saves a lot of programming to write similar rules with only small differences.

The information of a model's parameters in this research is stored in the data base and is represented as multiple-field facts. Each field of a fact , i.e., a character string or a number, has a different meaning and is separated by a space. Most facts have several common fields shared with each other, especially in this project, since the model parameters are categorized by only five types: fault location and dip , and bed depth, dip and density. These will be discussed in the following chapter. The fields of a fact can be bound by variables in the predicate. Thus, CLIPS can use the same predicate with several variables to match similar facts without any change. Furthermore, the facts already used can be removed from the data base if needed, and the facts reasserted in the execution portion are regarded as new facts. Therefore, using these special capabilities, CLIPS rules can be retriggered by matching other unused similar facts without any further change.

There are other significant features of CLIPS, such as conflict-resolution and multiple-field wildcard variables. A more detailed description of CLIPS can be found in the CLIPS manual or in Brooke's CLIPS software review paper (Brooke, 1988). A rule example to illustrate the capability of CLIPS in more detail is shown as follows:

```
    (defrule check-fault
         (constraint f ?t ?n ?v)
    ?s1<-(result f ?t ?n ?m ?c)
    ?s2<-(check-results)
    =>
       (retract ?s1 ?s2)
       (bind ?df (abs (- ?m ?c)))
       (if (<= ?df ?v)
        then
          (setpar f ?t ?n ?c)
        else
          (bind ?c1 (/ (+ ?m ?c) 2))
          (setpar f ?t ?n ?c1))
        (assert (check-results)))
```

As the rule name check-fault implies, this rule is used to check fault parameters. The predicates of this rule have some variables, beginning with a question mark, used to bind fields of facts that have "constraint f" and "result f" on the first two fields. There are two variables "?t" and "?n", which can bind either a character string or a number, used in both of the first and second predicates. This forces the "result" and "constraint" predicates to have the same values on the third and fourth field respectively, while these two variables are initially bound in the "constraint" predicate. The "result" and "check-results" facts are bound to variables "?s1" and "?s2" respectively, so that they can be removed from the data base and allow this rule to be used in multiple instances. When these three predicates are satisfied by the existing facts in the data base, then the rule will be triggered and executed. All of the first strings in the action portion are functional commands and followed by arguments. The built-in functions include "retract", "bind", "if", and "assert"; "setpar" is a user-defined function in this example. This rule first retracts the two bound facts from the data base, and then asserts the (check-results) fact again. This illustrates how the rule can be retriggered whenever a "result" fact exists and has the field values the same as the values bound in the "constraint" predicate, while the "constraint" predicate matches all of the candidates sequentially.

CLIPS satisfies all of requirements for an expert system tool needed in this project. The rule-based system has been created to control the FORTRAN inversion program and help it to find a solution compatible with the default or user-requested constraints. The algorithm for finding a solution using the rule-based system will be discussed in the following chapter.

IMPLEMENTATION

The overall goal of the project is to use the rule base of an expert system shell to implement geological constraints in gravity inversion. The technique used to achieve this goal to include FORTRAN inversion subroutines as user-defined functions in a CLIPS rule base. There are six major tasks in the construction of this program: construction of the inversion program, design of geological parameterization, linkage of CLIPS and FORTRAN, construction of the rule-based system, introduction of weighting parameters, and incorporation of geological constraints. The system was developed incrementally from a rudimentary prototype, and the various tasks interacted with each other during the implementation. These major tasks are discussed sequentially in the following sections.

Construction of the Inversion Program

The inversion program in this project is responsible for the mathematical calculation. It works with the rule-based system by taking the parameters or arguments passed from the rule-based system and returning the calculated results.

To reduce the task of programming, a two-dimensional gravity inversion program named GRV2D, written in FORTRAN, developed by the Center for Potential Fields Studies (CPFS, 1985) at Colorado School of Mines, was extensively modified. This inversion program uses the Marquardt inversion technique to compute the corrections for the parameters of a model, and uses a Talwani two-dimensional algorithm to calculate the gravity response of the model.

Originally, the GRV2D program was designed for an Apollo DN300 computer, while this project was developed on a Masscomp 5500, so there are a number of machine-dependent routines such as graphics routines which required modification. I modified the program to work on the Masscomp, but it will also run on other UNIX machines. I rewrote the graphics routines using VPLOT, a graphic package developed by Stanford University (Stanford, 1987). VPLOT is designed to be machine-independent; the graphics instructions are written into a metafile and then plotted by a machine-dependent interpreter. The modified program only keeps those routines from the original program which form the skeleton of the Marquardt inversion. Those subroutines include "inv2d", "stikp", "corbat", "splitp", "invers", "deriv", "rms", "lamda", "solve", "gelim", "augm", "printm", and "addm". Most of those routines were also expanded to fit the needs of this project. In addition, there are two FORTRAN routines from the CLIPS manual, "loadc" and "storec", used as the interface to convert string variables and symbols between FORTRAN and CLIPS. All of the FORTRAN routines, both the one newly created and those modified, are listed in Appendix A.

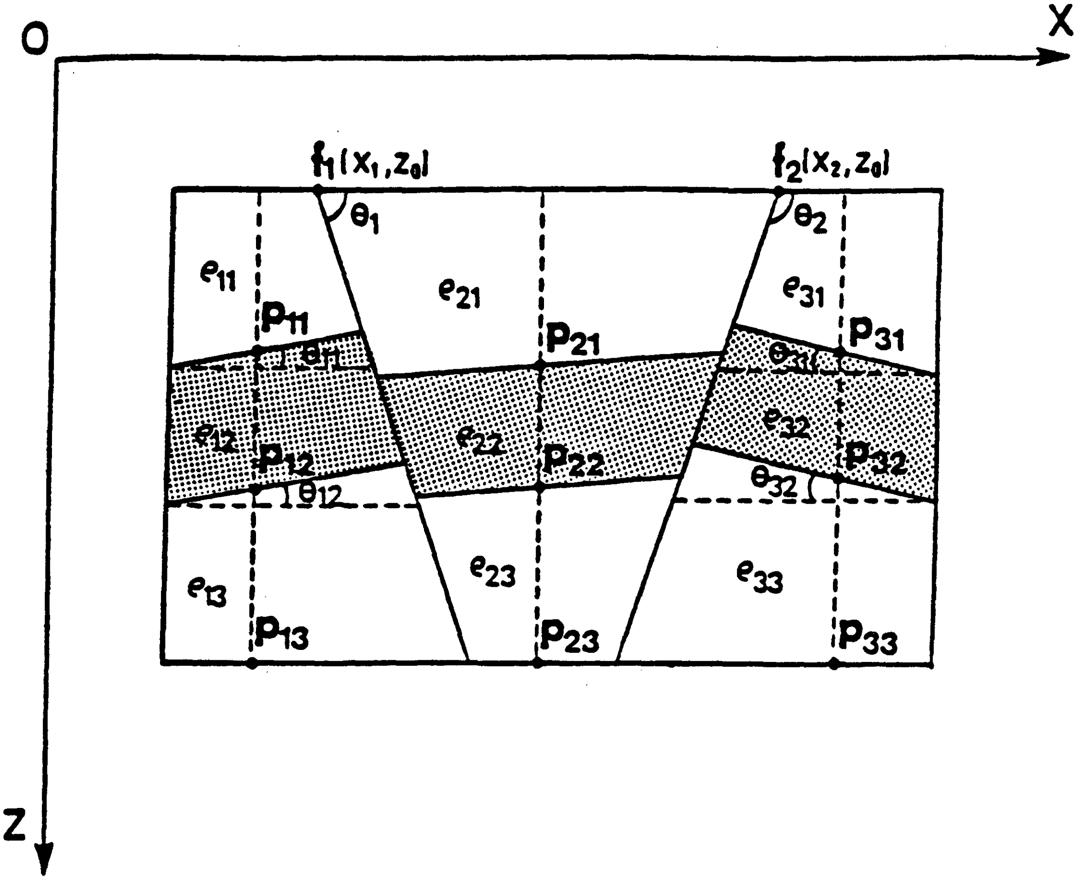
Design of Geological Parameterization

Most conventional inversion programs take the vertex coordinates of a model as the parameters, as does the program GRV2D, and can, in most cases, only deal with rather simple models. The vertex coordinates are only indirectly related to geology, and it is difficult to incorporate geological constraints using them. Consequently, a new parameterization algorithm was needed for this project to overcome the limitations imposed by the conventional parameterization, since the incorporation of geological constraints and the ability to deal with general multiple-source models were the most fundamental goals of this project. A parameterization using fault locations and dips, bed depths and dips, and densities was created to replace the vertex parameters which are used in conventional gravity modeling.

Figure 4 shows the new parameters used in this project. There are two types of parameters: fault parameters and bed parameters. The fault parameters include the fault plane (line) location (X_j , Z_j) , i.e., the intersection with earth's surface, and fault dip angle θ_j for the jth fault. The bed parameters include any point on the lower surface of the bed (X_{ij} , Z_{ij}) , the dip angle θ_{ij} of the bottom surface of the bed, and the density ρ_{ij} for the jth bed in the ith block.

This modified inversion program accepts a model which has any number of faults and beds and any dip angles. To simplify the calculation, I confined the outline of the model to be rectangular in shape, the top and bottom of model to be flat, and the X_{ij} , which can be taken as the location of a vertical borehole, to be constant for each block. Of course, it is still possible to expand the program to deal with any type of model.

The inversion program does not calculate the vertices of the source bodies directly; instead, it computes and corrects the geological parameters first, then transforms them into vertex coordinates using the Talwani technique to calculate the response of a model. The transformation of a model is straightforward. We can use the intersections of



fault parameter: $\begin{cases} \text{constant: } z_o \\ \text{variable: } x_j, \theta_j \end{cases}$

bed parameter: $\begin{cases} \text{constant: } x_{ij} \\ \text{variable: } z_{ij}, \theta_{ij}, \rho_{ij} \end{cases}$

P_{ij}: bed depth's point (x_{ij}, z_{ij})

ρ_{ij}: density

Figure 4. The geological parameters for a multiple-source model.

lines to calculate the vertices, since any source body is confined by several lines, and these lines are determined by the geological parameters. For example, the jth fault line can be expressed as

$$Z = Z_0 + (X_j - X) \tan \theta_j \qquad ;$$

the jth bed in the ith block line can be expressed as

$$Z = Z_{ij} + (X_{ij} - X) \tan \theta_{ij} \qquad ,$$

where (X , Z) is the vertex coordinate, (X_j , Z_0) and θ_j are fault parameters, and (X_{ij} , Z_{ij}) and θ_{ij} are bed parameters defined above. So, for each source body, we just need to calculate the intersection of the related lines to obtain the necessary vertices. However, since the Talwani algorithm is sensitive to the sequence of source body vertices, we also need to check the order of the vertices. These vertices are checked and ordered by the routines named "vertex" and "vertex1" prior to the call to the routine "corbat" to compute the gravity response of the model. The program calculates the response of one source body at one time, each block's contribution is accumulated from the bottom bed to the top. The total response of the model can be obtained by summing the contribution of each source body.

Linkage of CLIPS and FORTRAN

The rule-based system is written in CLIPS. CLIPS itself is written in C, but the inversion routines are written in FORTRAN. Therefore, designing the interface between CLIPS and FORTRAN is a crucial task in this project.

The inversion program was modified to work interactively with CLIPS rules. All of the FORTRAN subroutines, which are callable from CLIPS, are declared as user-defined functions in the C source file named "main.c" , and they are all listed in Appendix B. Basically, in CLIPS, the FORTRAN subroutines are called as external functions and used like the built-in functions in the THEN portions of rules. The parameters can be passed in both directions between CLIPS and FORTRAN. CLIPS can pass parameters to the FORTRAN subroutines in the format:

```
( <subroutine-name> <<arguments>> )
```

where <subroutine-name> is the name of FORTRAN user-defined subroutine, <<arguments>> are any bound CLIPS variables or literal values. In contrast, FORTRAN subroutines pass the parameters by asserting facts into CLIPS. These facts must be stored in character strings and the strings must be converted before calling the "assert" function. The facts assertion code can be found in the FORTRAN routine "invers" listed in Appendix A.

Because of the different methods used to store strings in C and FORTRAN, we have to use some special functions to convert C strings to FORTRAN and back. All of the string conversions are handled in FORTRAN. The non-numeric arguments passed from CLIPS must be converted to FORTRAN strings by calling the "loadc" function before being used in the FORTRAN routines, while the FORTRAN string data, used as facts in CLIPS, must be converted and passed to CLIPS by calling "storec" and "assert" functions. Compared to string conversion, passing a numeric argument between CLIPS and FORTRAN is much easier. The only complication is that CLIPS only takes and passes real numbers even if the number is an integer. This can be easily resolved by the use of float and integer conversion functions in FORTRAN.

Construction of the Rule-Based System

Construction of the rule-based system is an extremely different experience from FORTRAN programming. To build a rule-based system, it is not necessary to declare array dimensions, nor to set the entry point and end as in a

FORTRAN program; instead, many English words and symbols are used, and they are surrounded by parentheses. Basically, a rule-based system consists of two essential components: one is the rule base, the other is the data base. The rule base contains a list of rules separated by a blank line, while the data base contains the facts. A rule is composed of two requisite parts: one is the predicate, the other is the action. The execution of a rule depends on its predicates being matched by the facts in the data base. Thus, writing the rules and organizing the data base are both important to building a rule-based system.

In this research, I used CLIPS to write the rule base. As discussed in the previous Chapter, CLIPS is an English-like computer language, and its syntax is fairly easy. Every rule of the rule-based system consists of a rule name, followed by the IF predicates and THEN actions. For example:

```
(defrule rule-name
   (predicates)
=>
   (actions))
```

The rule name is a word, which can be composed of any valid symbolic atom, and can be made long enough to contain the meaning of the rule. For example, "enter-observed-data" is a real rule name used in this system, and it is clear enough from the name what the rule does. Therefore, it is very easy to recognize the function of each rule in this system by the rule name. This saves much effort in adding further comments to each rule.

Essentially, the rule base plays the following roles in this project: first, it controls the logical flow of the system; second, it embeds knowledge about the gravity modeling technique; and third, it incorporates geological constraints.

A flow chart shown in Figure 5 represents the general structure of the rule-based system in this research. There are several rules to implement each step shown in Figure 5, and they are named in a similar way to the step names. For example, the step "enter-data" includes the "enter-observed-data" and "enter-initial-model" rules , "check-results" includes the "check-fault", "check1-layer", and "check2-layer" rules and so on. The computer code of the rule-based system is listed in Appendix C. The main function of each rule can be easily recognized by its rule name.

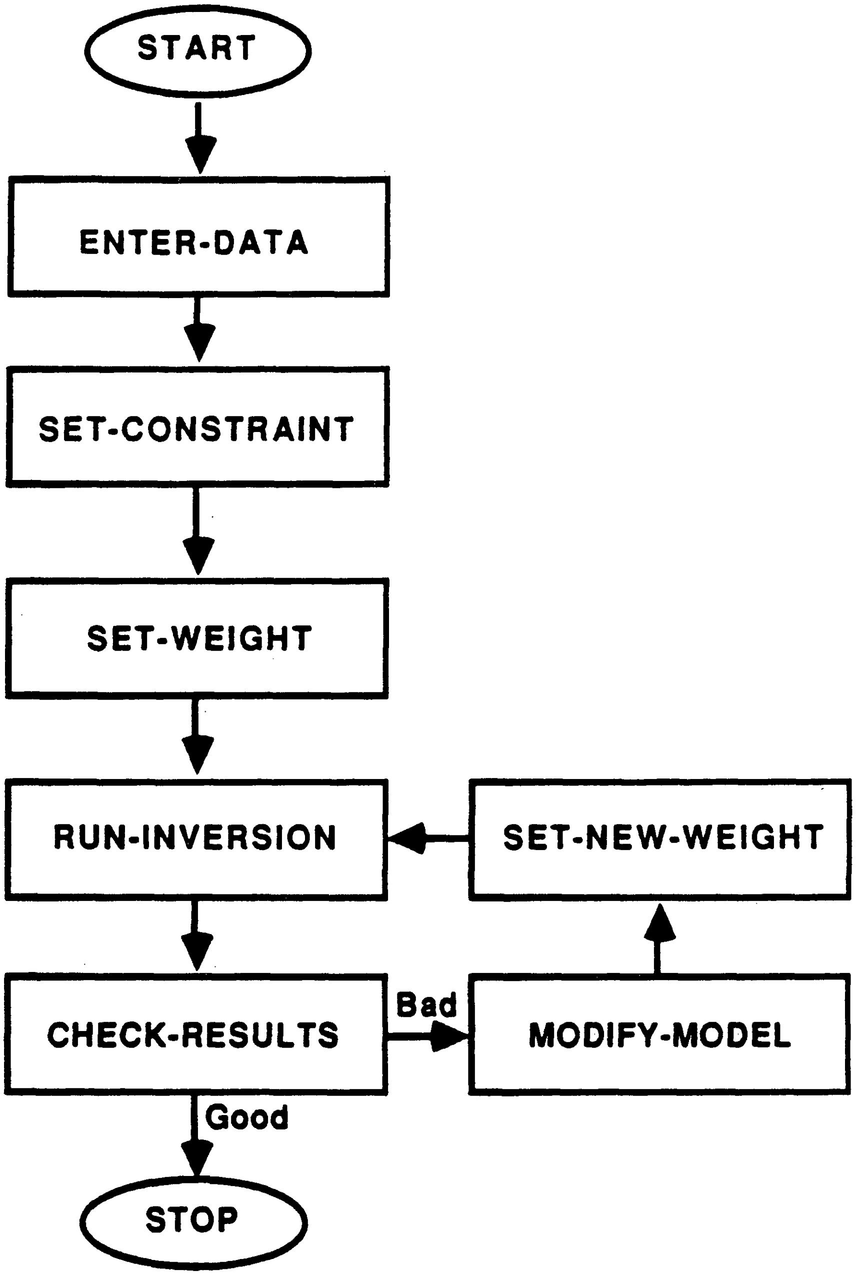


Figure 5. Flow chart of the rule base.

This rule-based system was updated gradually from a rudimentary prototype which had been created in the beginning stage. Some important policies and critical ideas were generated during the implementation because of unexpected events, such as assigning weights to parameters to resolve the unreasonable correction ranges. The use of weighting factors and incorporation of the geological constraints will be discussed later in this Chapter.

Weighting Parameters

A set of gravity data taken from the National Petroleum Reserve in Alaska (NPRA) was used to test the program while the basic structure of the rule-based system was being constructed. Because there is a lack of subsurface information, I assumed that there are two low-density intrusions, with density contrast -0.10 g/cm^3 , which correspond to the gravity lows on the profile shown in Figure 6, in which the observed data is shown as a dashed line, and the model response is shown as a solid line. The Root Mean Square Error (RMSE) between the observed data and the modeled values is about 13 mGal. The model calculated without constraints, shown in Figure 7, has a RMSE of about 1.8 mGal.

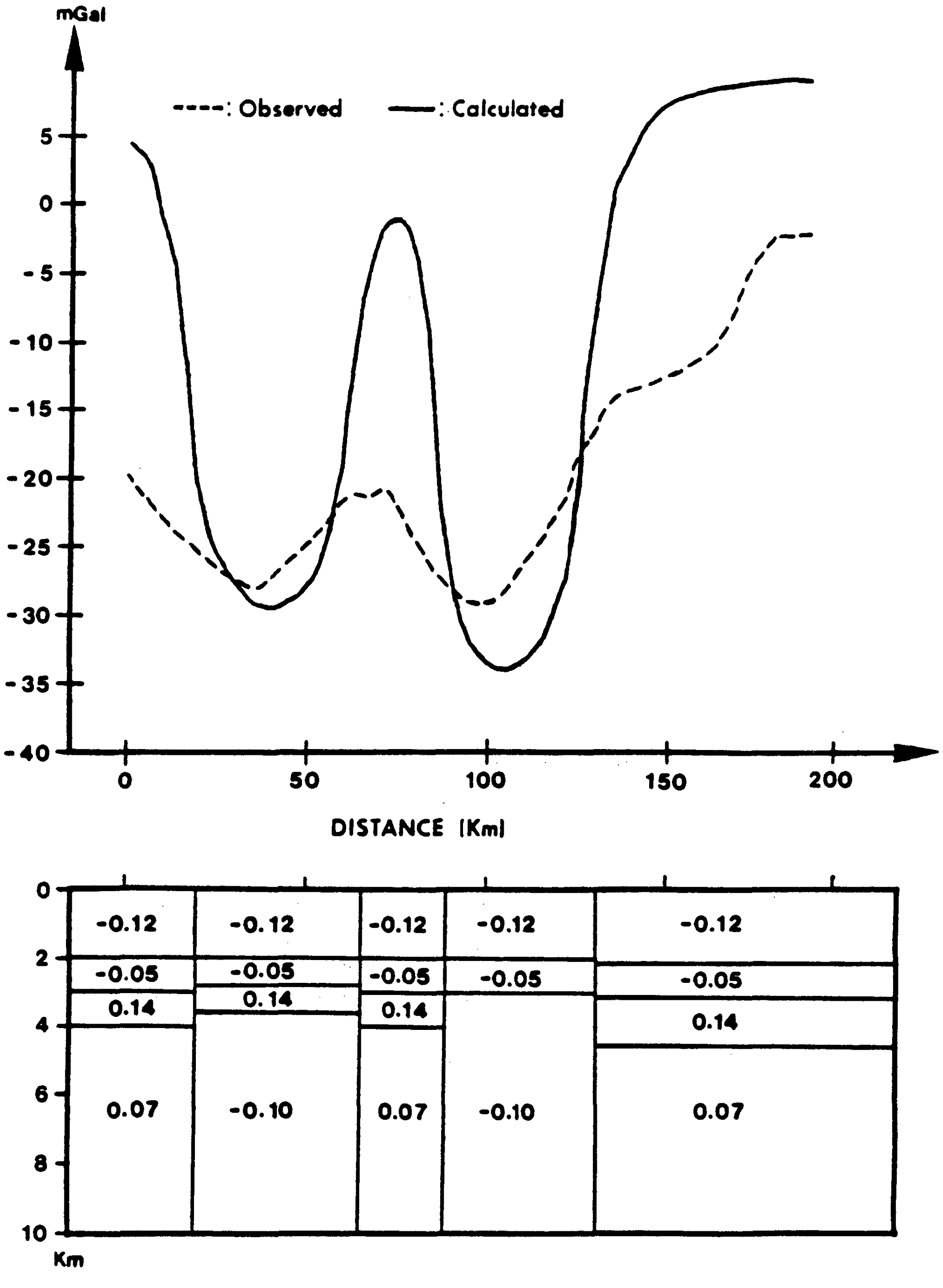


Figure 6. Gravity anomaly profile from the National Petroleum Reserve in Alaska (NPRA) and the initial model response.

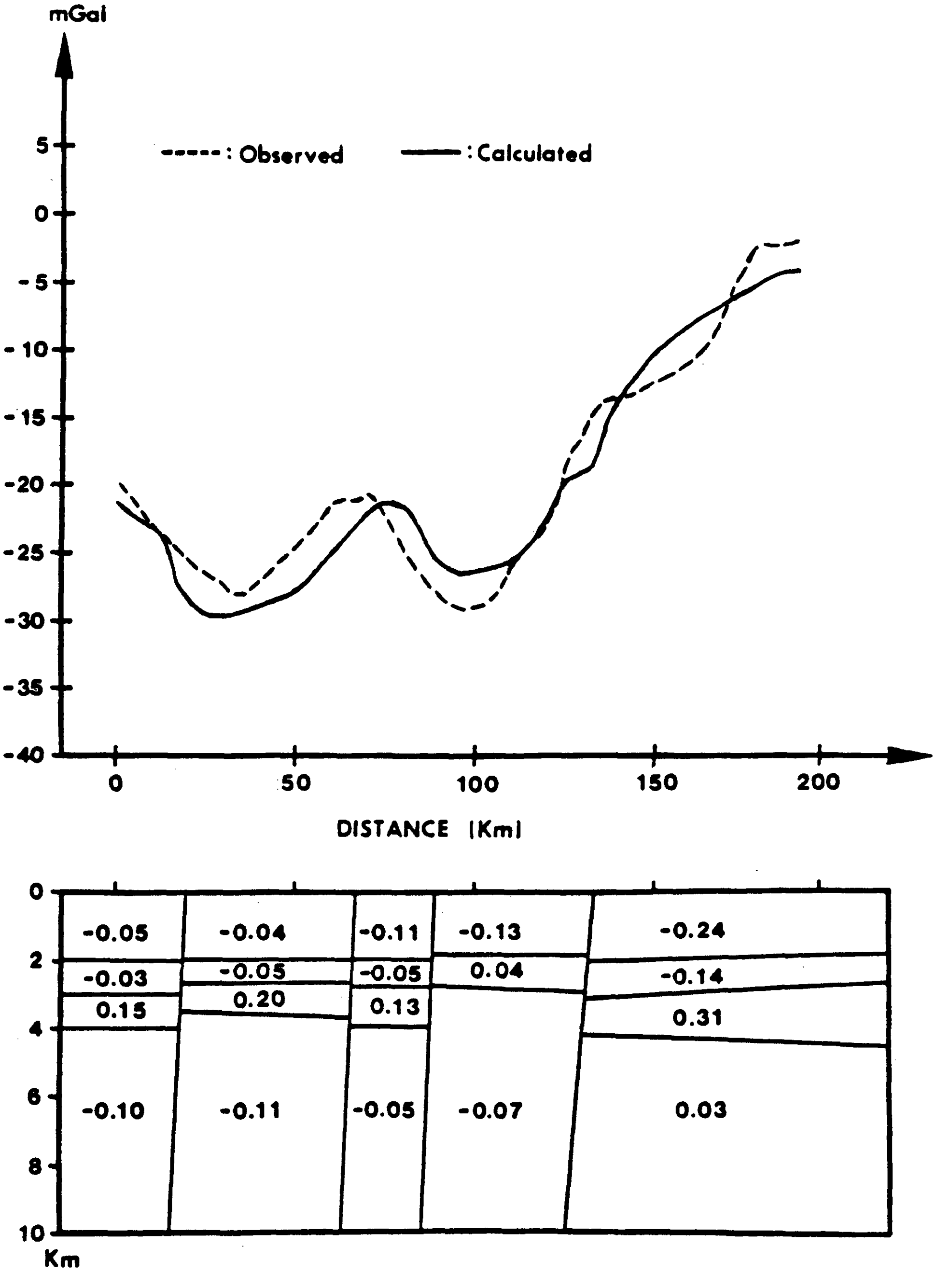


Figure 7. Model response calculated without constraints corresponding to the gravity anomaly shown in Figure 6.

From this example, it is clear that the inverse computation in this system converges very well, but that there are some problems to be resolved. For example, the changes in the density contrasts are much larger than those in the rest of parameters; one source density contrast at the lower left corner is changed about 250% from +0.07 to -0.10 g/cm^3, while the parameters other than density contrasts are changed by less than 5%. This result is unreasonable.

I examined the matrix A^TA , where the matrix A consists of the first derivatives of the model response at each observed point on the surface with respect to the parameters of the model, the matrix A^T is the transpose of matrix A , and A^TA is the product of A^T and A. I found that the elements of A^TA related to the density parameters are much larger than the other elements. Thus I designed a strategy for improving the balance, by assigning a different weighting factor for each parameter to confine the correction for that parameter within a reasonable range. Thus the contents of the matrix A are changed to

$$A_{ij} = \frac{\partial M_i}{\partial P_j} * W_j$$

where M_i is the gravity response of current model at the ith observation point, P_j is the jth parameter, and W_j is the weight for P_j . In addition, the correction term of each parameter is changed to:

$$P_j^{new} = P_j^{old} + \Delta P_j * W_j$$

Thus, the magnitude of corrections can be totally controlled by the weighting factors, and unreasonable corrections avoided. There are at least two other advantages of the use of weighting factors: first, the weights can be used to enforce the constraints; second, the weights can be adjusted according to human experience. That is to say, the weighting factors can be used to implement geological constraints by setting them to suitable values based on the allowable correction range. For example, a parameter which has a broad allowable correction range can be given a larger weight, while one with a narrower allowable range can be given smaller weight. Thus, the weights can be used to control the magnitude of correction for every parameter based on the given allowable ranges or the user's expectations. The use of the weighting factors to implement the constraints will be discussed in detail in the following section.

Incorporation of Geological Constraints

The rule-based system was first designed only to check the inversion results, but was then expanded to incorporate constraints such as continuity of layer densities, continuity of fold boundaries, constant fault throws, etc. Since the weighting factors were used to control the magnitude of the parameter corrections, the rule-based system was then further expanded to help the inversion to optimize the gravity model in a more geological sense by adjusting the weighting factors appropriately. The final rule-based system is designed to be not only capable of checking if the mathematical calculation is satisfactory, but also to be able to help the inversion to generate a geologically reasonable model.

Every parameter of a model used in this research is an element of geological information, such as fault location, fault dip angle, bed density, bed depth, or bed dip angle. All of these elements can be easily integrated with the geological constraints, i.e., any subsurface information known from other geophysical or geological studies related with the above parameters can be used as the constraints and embedded into the program to ensure that calculated results are compatible with them.

All of the parameters used in this program are assigned an individual weighting factor; these weights are initially given by defaults in the rule base. Therefore, the flexibility of each parameter is fully controlled by the weight, and its updated range can be altered by simply changing the weights. That is to say, if the calculated correction for the parameter is too large to be accepted, then its weight would be reduced automatically in the next computational run; on the other hand, if the correction is too small compared with the allowable range then the weight would be enlarged in the next run.

The weights for the constrained parameters are automatically changed during the inverse processing until the their corresponding corrections are acceptable, while the weights for those unconstrained parameters are also changed, and randomly multiplied by a factor slightly different from 1. This is to avoid the inversion calculation being occasionally trapped in a loop without any improvement on the current model. If the same initial model is input to the inversion program, it will return the same result under the same circumstance. Therefore, if the inverse results are not satisfactory, then the rule-based system should not send the same initial model into inversion again. Thus, the rule-based system will modify the calculated model to keep the satisfactory parts unchanged, while the unsatisfactory parts would be changed based on the allowable correction ranges as well as reducing or increasing the related weights, and the weights of unconstrained parameters will be slightly changed by multiplying by a factor in the range of 0.8-1.2.

This system also provides for a layering constraint different from the parameter constraints mentioned above; that is, separate source bodies belonging to a single layer can be constrained to have the same density during the calculation. For example, in a layered model, one layer may be parameterized by several different blocks, and their parameters are independent of one another and are updated differently by the inversion. However, they should have the same density. The programming strategy for implementing this constraint is to enforce those layer-related bodies to take the same correction value to update their density parameters; the resulting value is the average of the respective corrections. Since this is a mandatory constraint, it is appropriate to implement it in the FORTRAN portion. Thus, this layer-density constraint is done in the FORTRAN subroutine "addm" prior to sending the updated result to the rule base.

Another mandatory constraint is to confine a fault to have a constant throw in the calculated model. This is useful for constraining a fault which has a well-known constant vertical displacement along the fault plane. A fault which has different throws in different sections can also be handled in this program; this case will be discussed later. This constant-throw fault constraint is simply implemented by enforcing the bed depths of the right-hand side of the fault to take the given displacement distance from the corresponding beds of the left-hand side.

The constant-throw constraint can also be used in another way to ensure the continuity of the layer boundaries for a fold model, in which the fault line is actually a fold axis. This constraint can be simply implemented by assigning the throw value to be zero, i.e., requiring that there be no displacement along the fold axis.

Another significant capability of the program is that it allows the user to keep parts of the model unchanged in the inversion. This is simply done by setting the relevant weights to be zero. Recall the parameters are updated by the equation:

$$P_j^{new} = P_j^{old} + \Delta P_j * W_j$$

That is to say, if the weight W_j is zero, then the jth parameter will be kept unchanged throughout the inversion calculation. Thus, in this rule-based modeling system, the weights are used not only to determine the flexibility of the parameters, they are also used to stop updating particular parameters. In this way, a fault with various throws for different sections mentioned above can be handled by weighting the related parameters to be zero, i.e., keeping the related bed depths and dips on the both sides of the specified fault unchanged. This can be used as a tool to determine which portion of the model gives rise to the essence of the observed anomalies, and can also be used to invert the model part by part if necessary.

Final Structure of the Program

The rule-based modeling system uses the inversion technique to search for the best-fit model, and uses the rule-based system to constrain the inversion computation geologically. This project not only creates a new research field of the application of inversion techniques, but also offers a useful two-dimensional gravity modeling tool.

As Figure 8 shows, this rule-based modeling system, named GMES, can invert a set of gravity anomalies to obtain

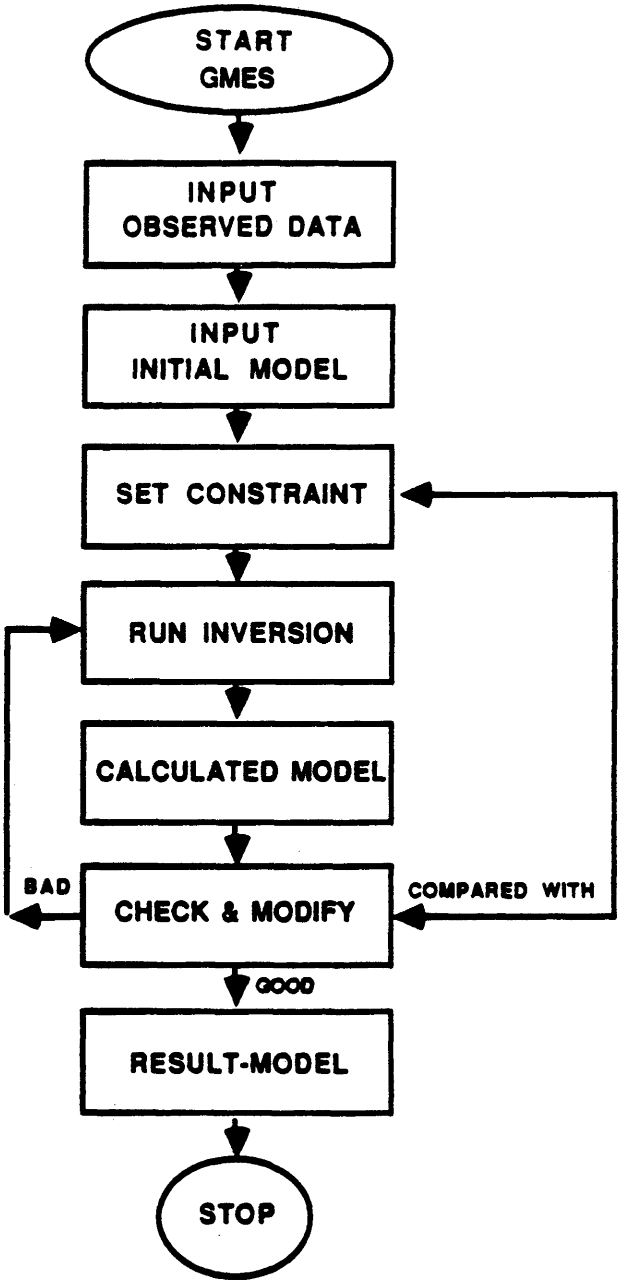


Figure 8. Flow chart of the rule-based gravity inversion modeling.

a best-fit and constraint-satisfied model based on the given initial model and constraints. Basically, there are two different loops in the program to optimize the model: one is in the inversion which is written in FORTRAN, and the other is the rule base which is written in CLIPS. The inversion iteration updates the model mathematically to fit the observations, while the rule-based iteration updates the model geologically and then feeds the updated model back to the inversion with different weights.

A variety of constraints can be embedded in the program, such as the maximum range of each parameter, continuity of layer boundaries and densities, and fault throws. The program also provides the capability to keep part of the initial model unchanged or changed within a reasonable range based on information from other geophysical surveys or the user's expectations. Generally, the more constraints embedded, the more difficulty the the system has in obtaining a model and the longer the computation time needed. If a large number of difficult-to-satisfy constraints are used, the system will have difficulty in finding a compatible model for the user. Therefore, the rule-based system was expanded to handle this situation by returning messages to the user and asking for relaxation of the relevant constraints for further runs of the program.

This makes the system work more realistically and more intelligently.

TESTING AND RESULTS

Two types of testing of this rule-based inverse modeling program were performed. The first set of tests used synthetic data with both a simple graben and a syncline model. The main purpose of these tests was to test all facilities provided by the program by integrating the various constraints. The other tests used a set of gravity data from Albuquerque basin, New Mexico, to demonstrate the capabilities of the program as a tool in gravity modeling.

Synthetic Data Testing

Synthetic gravity data, shown as a dashed line in Figure 9-a, was used as the observed data for the synthetic data tests. A simple graben model, also shown in Figure 9-a, was used as the initial model in the first few tests. The RMSE between the observed data and the initial model response is 12.134 mGal. The initial model response is quite similar in shape to that of the observed data. This model was intended to demonstrate the basic facilities of the program. The differences between the calculated model and the initial model were emphasized by making changes in the densities in the following tests. All of the RMSE limits were set to be 1 mGal in the tests discussed in this

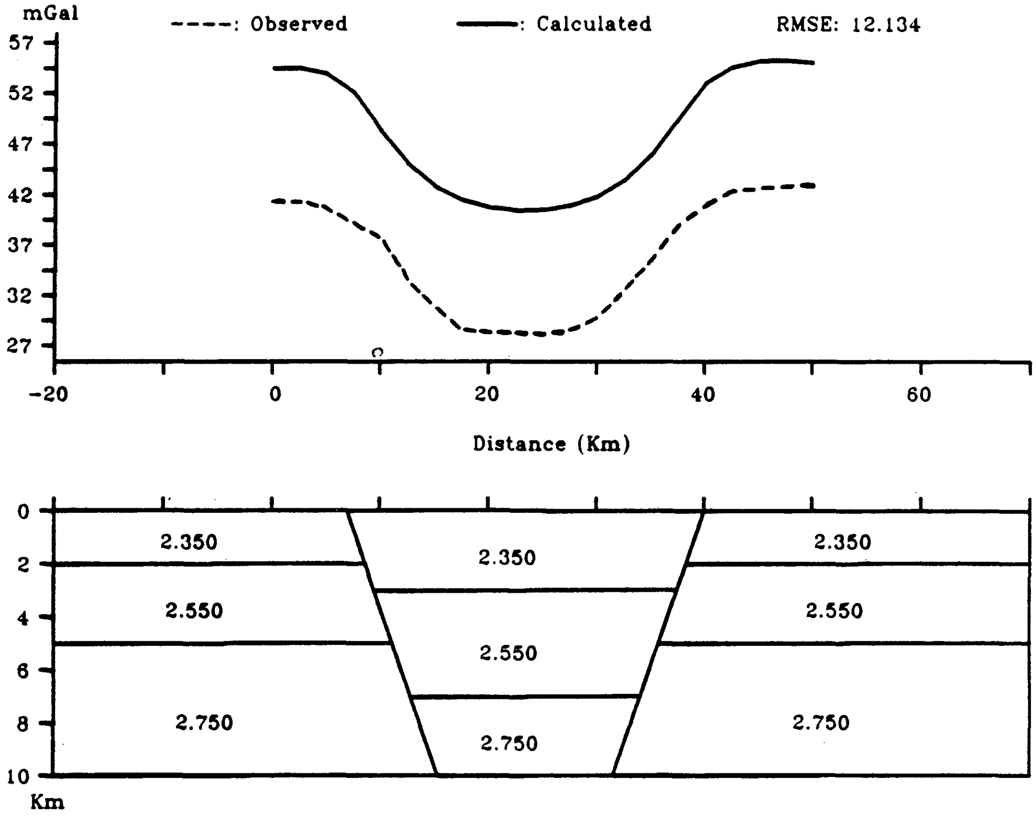


Figure 9-a. Synthetic gravity anomaly and the initial graben model used in Figures 9-b to 9-f.

section. The mean density is 2.45 g/cm^3. The program was tested using various constraints such as allowable maximum range for each parameter, zero change for certain parameters, continuity of layer densities, and constant throws, respectively. In addition to testing the program using the graben model, I used the same synthetic gravity data to be modeled as a fold model.

Figure 9-b shows the calculated result without constraints. The program returned this unconstrained model in a few seconds and iterated only once. As mentioned above, the changes in the density parameters are larger than those in the other parameters. The following density changes were observed: the densities of the topmost layer were changed from 2.35 to 2.517, 2.520, and 2.528 g/cm^3 respectively, while those of second layer were changed from 2.55 to 2.36, 2.235, and 2.323 g/cm^3 respectively. The trend of the density distribution was reversed from increasing downward to decreasing downward between these two layers. From this result, it is apparent that the unconstrained inversion calculation may return artifacts which are mathematically satisfactory but geologically unreasonable.

Figure 9-c shows the calculated result in which the density, depth and dip parameters of the topmost layer of

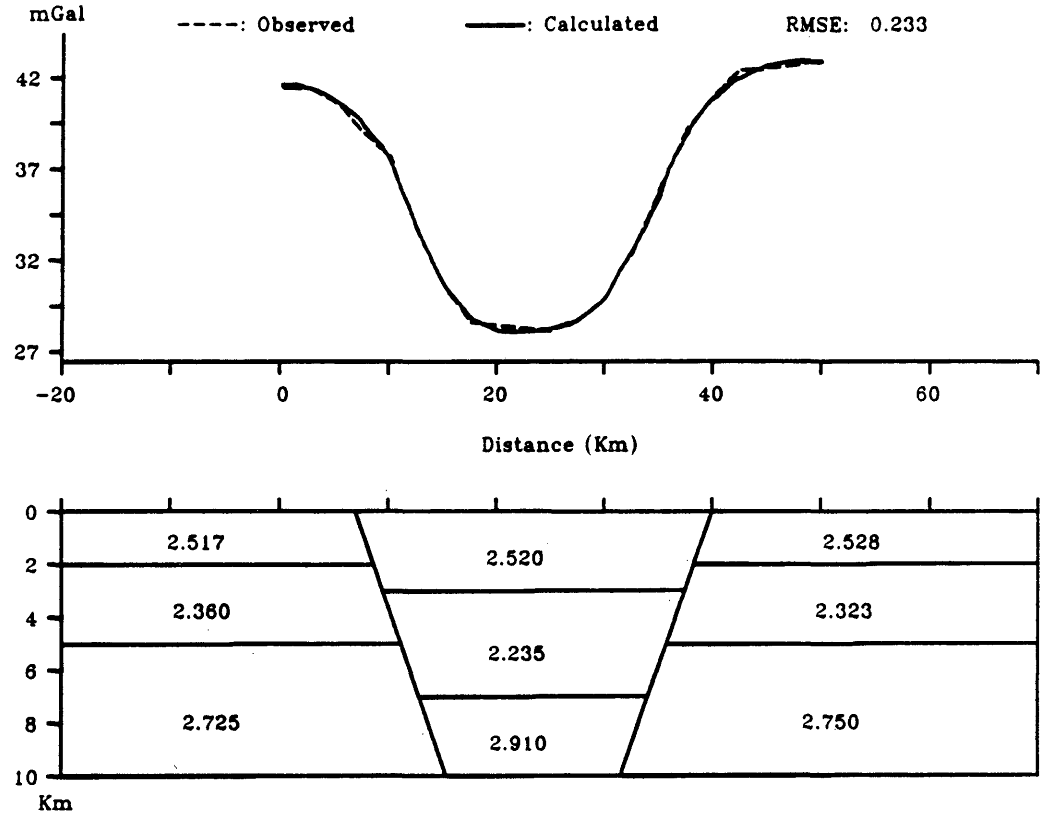


Figure 9-b. The calculated model based on the initial model shown in Figure 9-a without constraints.

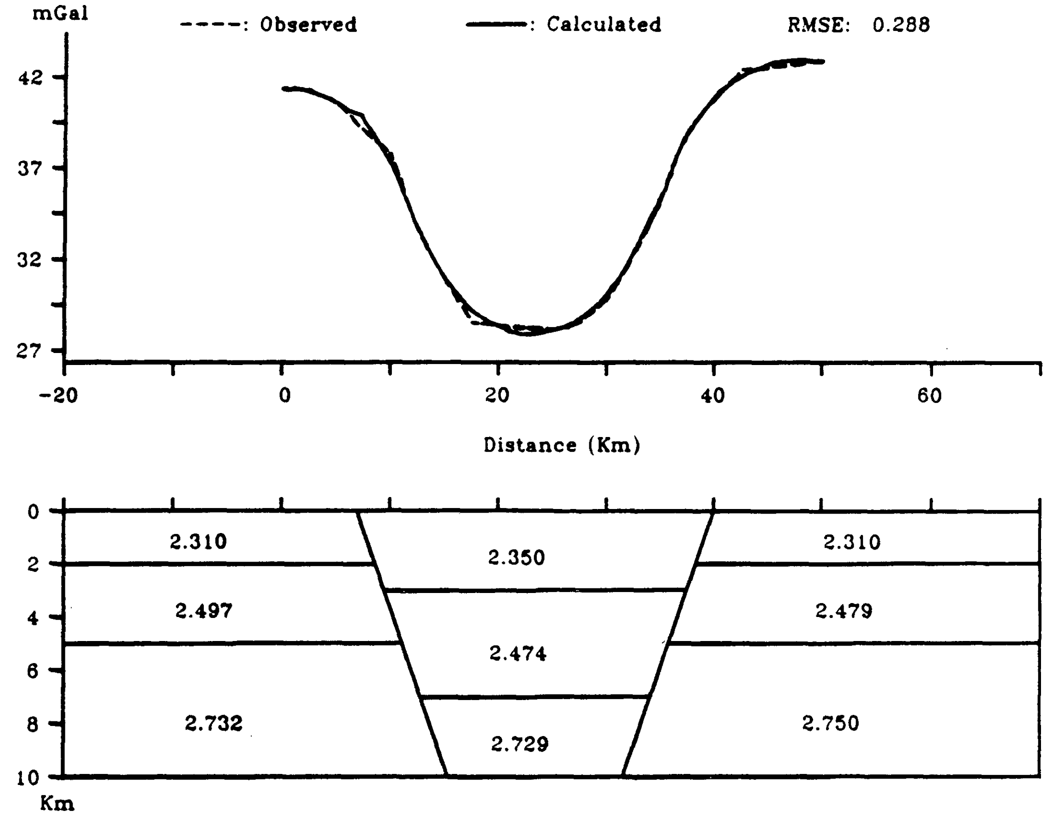


Figure 9-c. **The calculated model based on the initial model shown in Figure 9-a constrained by fixing the parameters of the middle-top block.**

the second block were kept unchanged, i.e., those parameters were constrained with a zero allowable correction range. This model was returned very fast too, but took two iterations. In this calculated model, the topmost block in the column has the same density value, 2.35 g/cm^3, as that of the initial model; in addition, the depth and dip parameters are kept unchanged as expected. This shows that the program can keep parameters unchanged through the inversion calculation. This facility can be used to keep parts of the model unchanged if needed.

Figure 9-d shows the calculated result in which the densities of the topmost layer were confined to be within a 0.01 g/cm^3 allowable correction range, as well as keeping the three different bodies of this layer at the same density. The calculated densities of the constrained layer were changed from 2.35 to 2.348 g/cm^3, a change of 0.002 g/cm^3 - within the required range. Furthermore, this constrained result has improved the problem of the unreasonable reversal in the trend of the density distribution found in the earlier unconstrained test. From the above tests, Figures 9-b and 9-d, we also find that different results can be obtained under the unconstrained and constrained circumstances using the same data set and the same initial model.

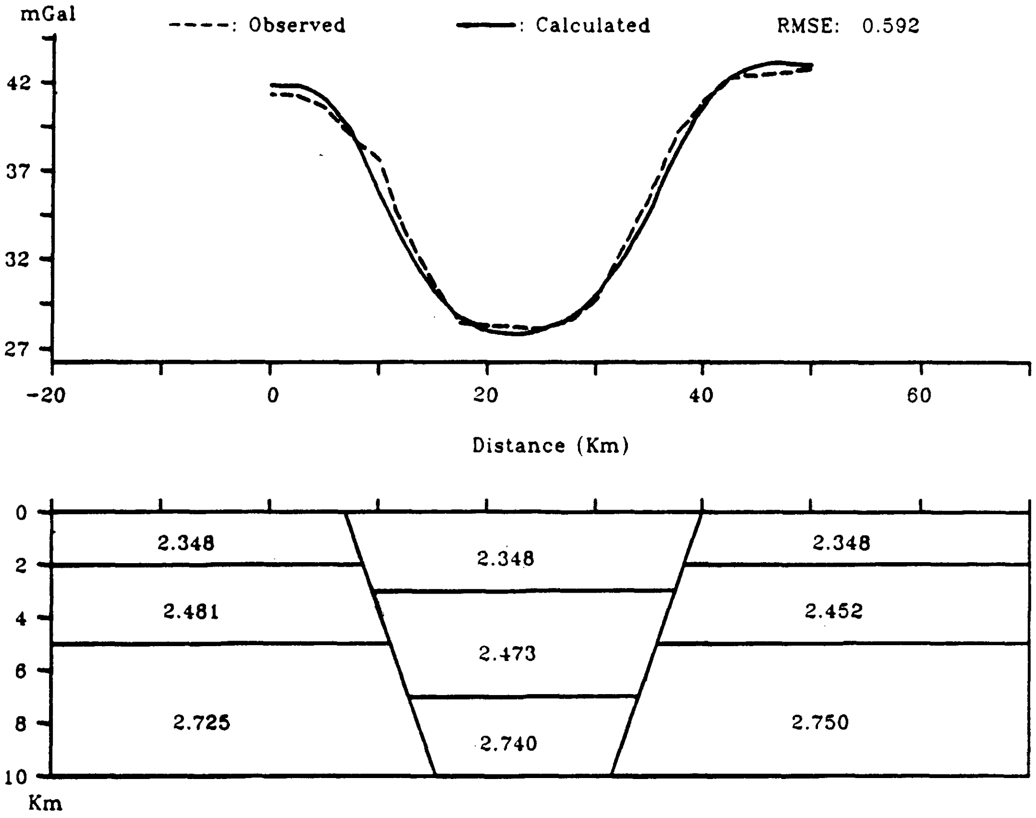


Figure 9-d. The calculated model based on the initial model shown in Figure 9-a constrained by setting the allowable correction range on the topmost layer to 0.01 g/cm^3 and keeping that layer at a constant density.

Figure 9-e shows the calculated result when each layer was given a constant density; the layer densities increase with depth. After inversion, the densities of the topmost layer were decreased from 2.35 to 2.399 g/cm^3; those of middle layer were decreased from 2.55 to 2.473 g/cm^3 ; and those of the bottom layer were decreased from 2.75 to 2.709 g/cm^3. The greatest difference between the initial densities and the calculated densities is 0.077 g/cm^3, which occurred in the middle layer. This result shows that the program can handle a model in which the densities are constrained by layering.

Figure 9-f shows the calculated results in which both faults were confined to have 1.5 kilometers of vertical displacement and each layer was kept at a constant density. The calculation has changed the thickness of the middle layer of the middle block to meet the constant-throw constraint, such that the depth of the first layer is increased while the second layer's depth is decreased, and the densities were increased from 2.35 to 2.366 g/cm^3 for the topmost layer, decreased from 2.55 to 2.483 g/cm^3 for the second layer, and decreased from 2.75 to 2.716 g/cm^3 for the bottom layer. This model took longer to calculate, about 4 minutes on the Masscomp, and the intermediate calculated models were sent to the rule base and back three times.

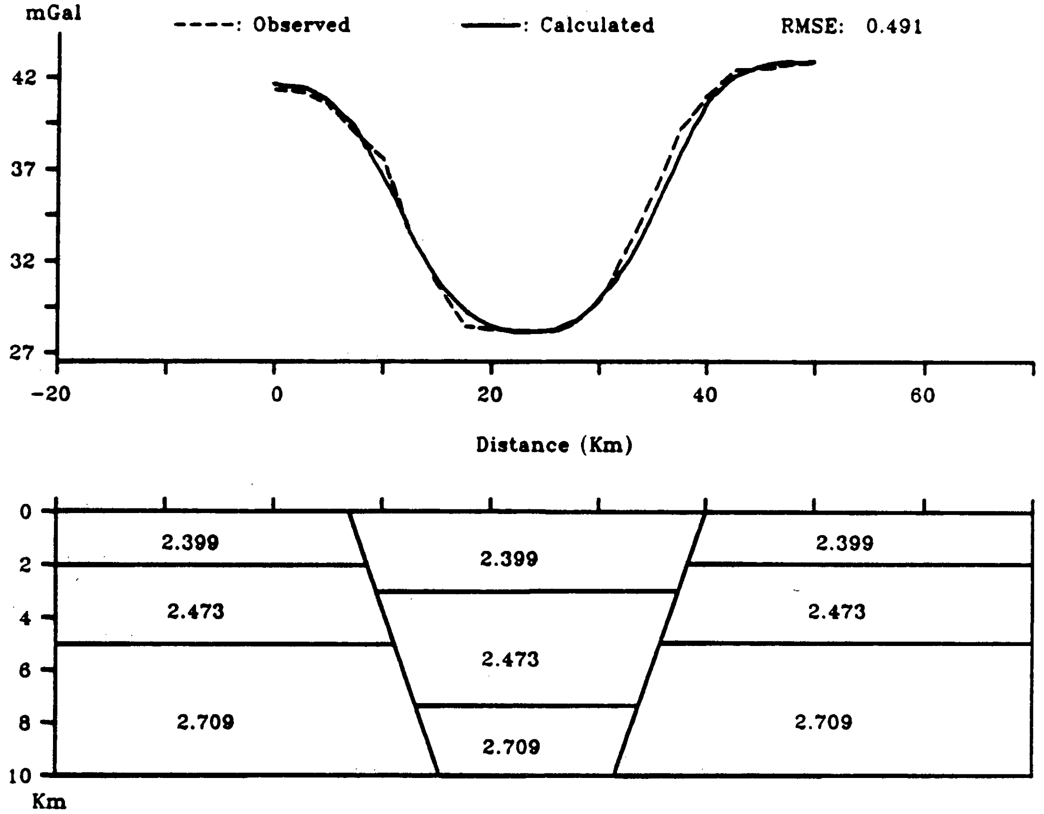


Figure 9-e. The calculated model based on the initial model shown in Figure 9-a constrained by keeping each layer at a constant density.

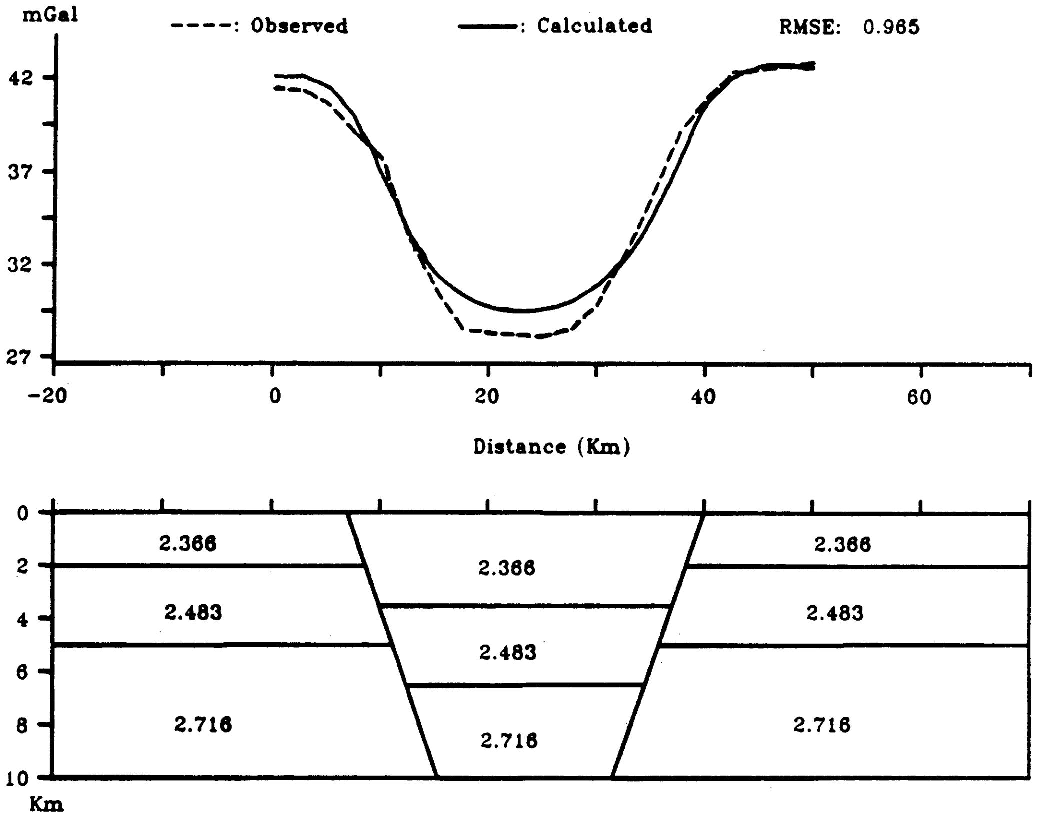


Figure 9-f. The calculated model based on the initial model shown in Figure 9-a constrained by keeping each layer at a constant density and setting the maximum vertical displacement on both faults to 1.5 km.

In addition to testing the program using the graben-horst model, this program was also tested using a fold model with the same set of synthetic gravity data. The folding model and its response are shown in Figure 10-a. The RMSE between the initial model and the observed data is 7.603 mGal.

Figure 10-b shows the calculated results in which the three fold axes are constrained to have zero throw and each layer is kept at the same density. The densities of the syncline model were increased from 2.35 to 2.457 g/cm^3 for the topmost layer, decreased from 2.55 to 2.543 g/cm^3 for the middle layer, and decreased from 2.75 to 2.638 g/cm^3 for the bottom layer, respectively. Furthermore, the bedding planes of the folded layers were all connected together as required.

From the above tests, we find that the program works quite well in finding a model which fits the data and satisfies the constraints. It not only can calculate a gravity model much faster than the trial and error method, but also can incorporate a number of different constraints. In the next section, the program will be tested using a set of real gravity data and a much more complex geological model.

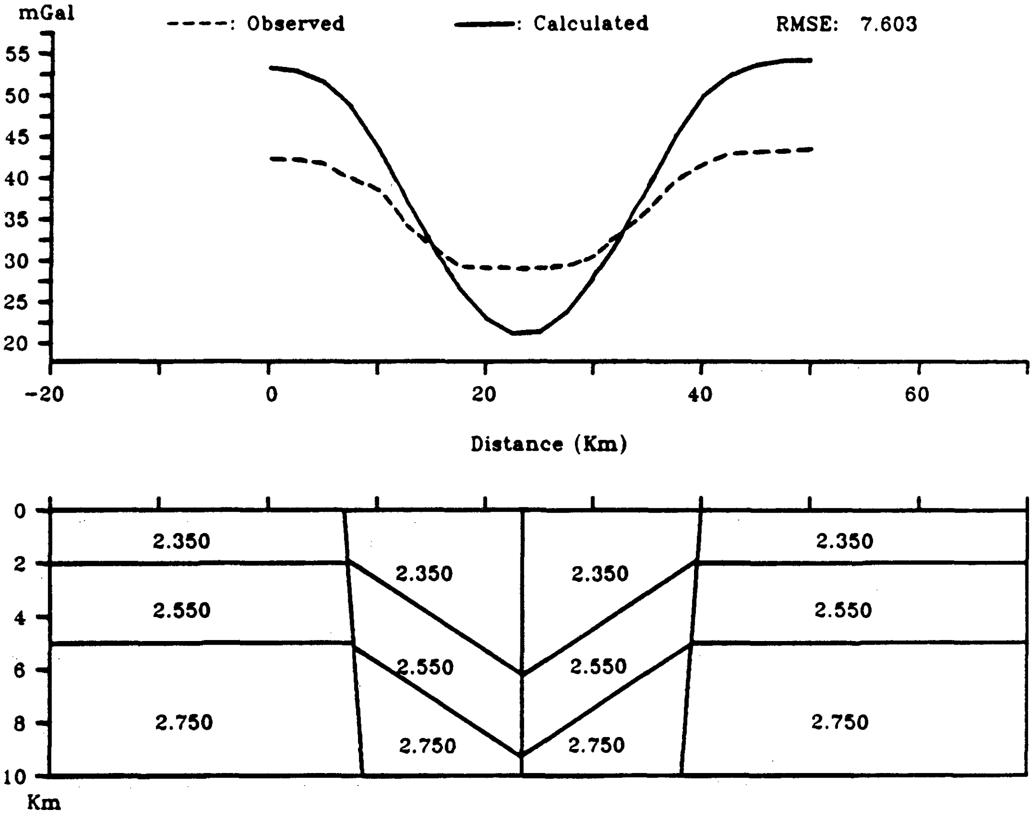


Figure 10-a. Synthetic gravity anomaly with an initial syncline model.

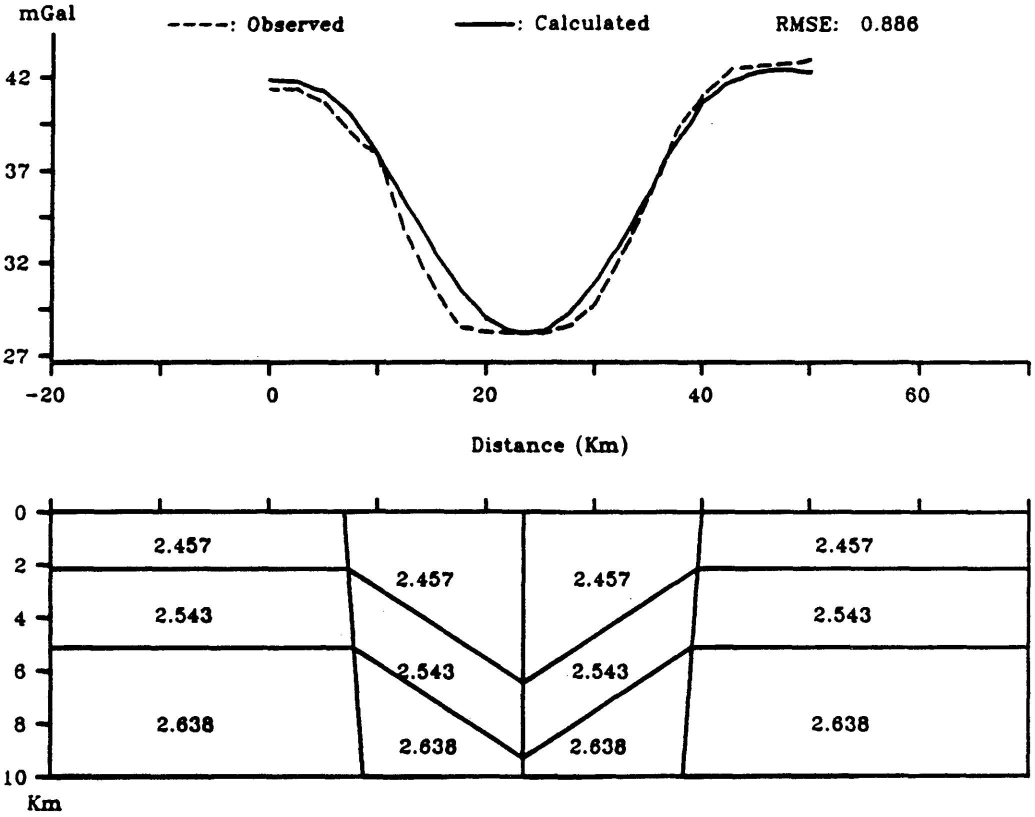


Figure 10-b. The calculated model based on the initial model shown in Figure 10-a constrained by keeping each layer at a constant density and forcing the three fold axes to have zero throw.

Real Data Testing

A set of gravity data which covers the Albuquerque basin in New Mexico (Birch, 1982), was used to further test the program. The Albuquerque basin is a portion of the Rio Grande rift (Kelley,1977). This basin appears as the rock unit S in the tectonic map shown in Figures 11-a and 11-b. It consists of sedimentary graben fill which is middle Miocene through Holocene in age. The basin is bounded on the west by the Colorado plateau and on the east by the the eastward-tilted Sandia, Manzano, and Los Pinos fault blocks (Kelley, 1977). From Figure 11-a, it can be clearly seen that both the west and the east sides of the Albuquerque basin, especially on profile A-B (Figure 12), are bounded by a number of normal faults, and there is a strip of intrusive rocks along the east margin of the basin. These intrusive rocks consist of acidic intrusive rock, schist and gneiss, and greenstone, which are believed to have higher average density than the granitic bedrock, whose density was taken to be 2.67 g/cm^3. The structures on the east side are quite different from those of the west side, and this gives the entire basin a marked asymmetry (Figure 12).

The Albuquerque basin appears as a gravity low in the complete Bouguer map (Figure 13). Gravity profile #1 shown

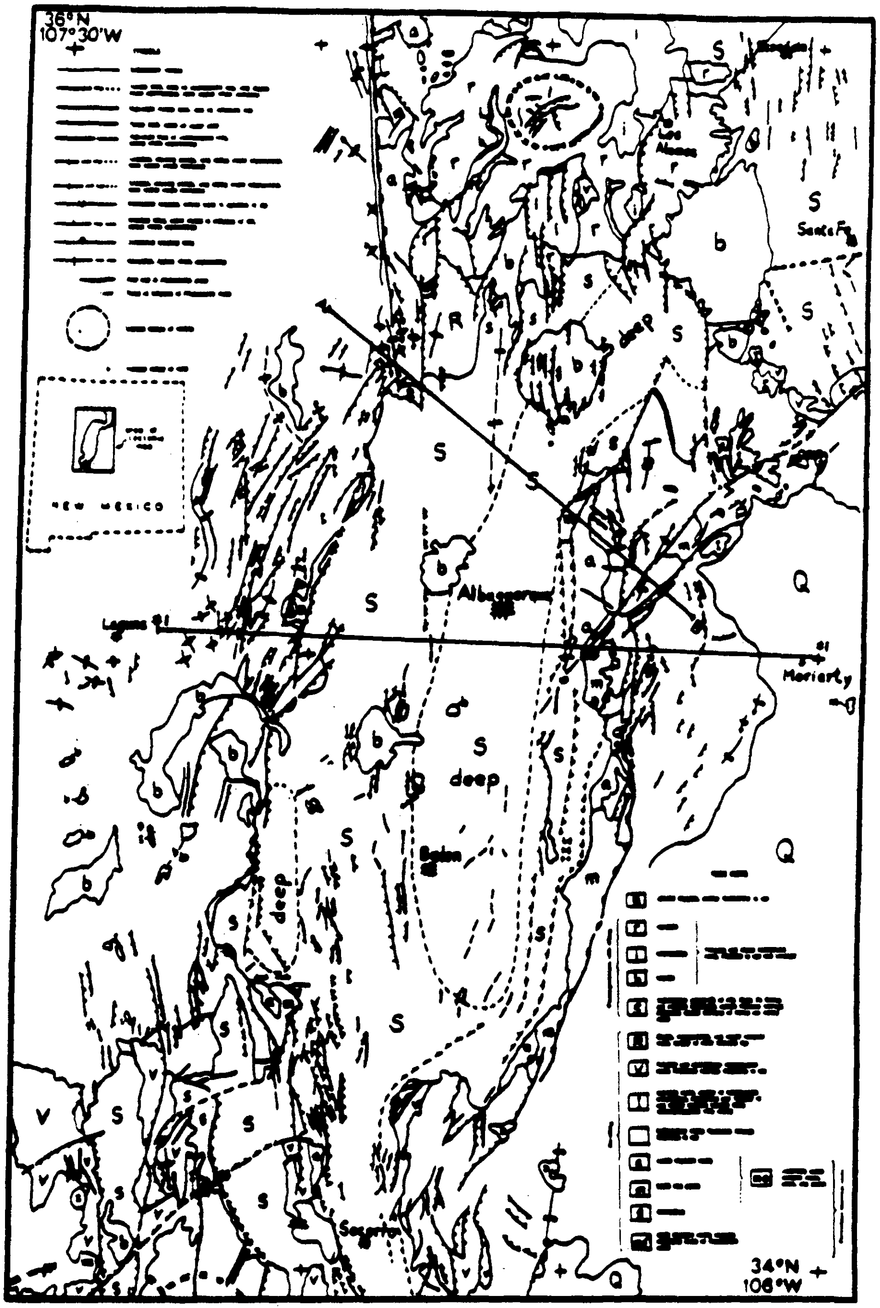


Figure 11-a. Tectonic map of the Albuquerque basin. Rock units and symbols are shown in Figures 11-b and 11-c respectively. (modified from Woodward et al., 1975).

ROCK UNITS

	Q	Alluvial deposits, mostly Quaternary in age			
Contemporaneous with rifting	r	rhyolitic		Volcanic and minor hypabyssal rocks, Pliocene in age and younger	
	i	intermediate			
	b	basaltic			
	S / deep	Sedimentary graben-fill of rift; Santa Fe Group and younger sediments; middle Miocene through Holocene; deeper portions of basins are darker shade			
Pre-rifting	R	Clastic, volcaniclastic, and minor volcanic rocks, mostly of early Miocene age			
	V	Volcanic and subordinate volcaniclastic rocks; Paleocene through Oligocene in age			
	I / =	Intrusive rocks, mainly of intermediate composition, Cretaceous and Tertiary in age; locally includes basaltic dikes contemporaneous with rifting			
		Sedimentary rocks; Cambrian through Oligocene in age			
	a	Acidic intrusive rocks	m, a	Undivided acidic intrusive rocks, schist, and gneiss	Precambrian basement rocks
	m	Schist and gneiss			
	g	Greenstone			
	mf	Mafic intrusive rocks; includes ultramafic rocks of Nacimiento uplift			

Figure 11-b. Rock units used in Figure 11-a.

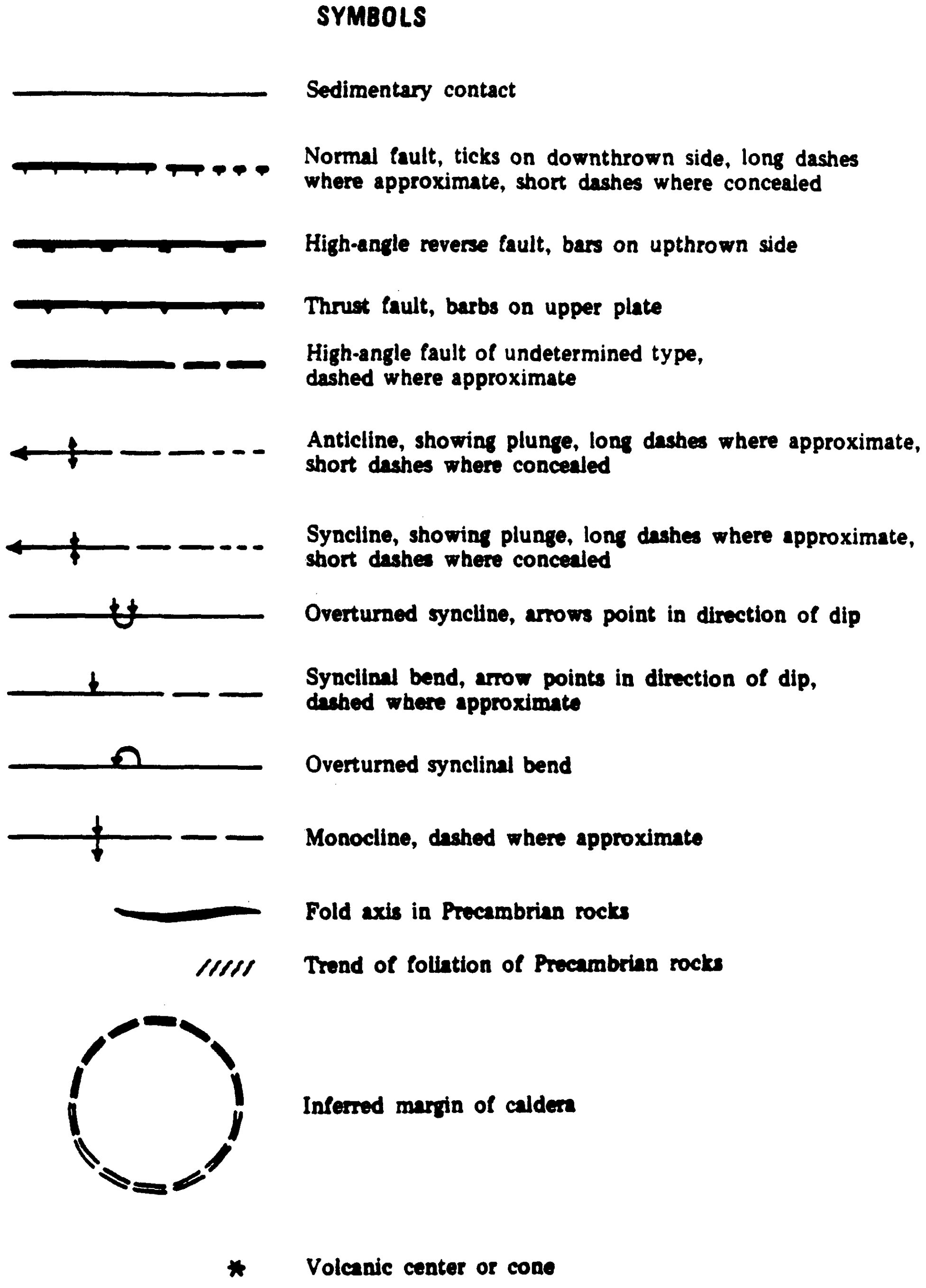


Figure 11-c. Symbols used in Figure 11-a.

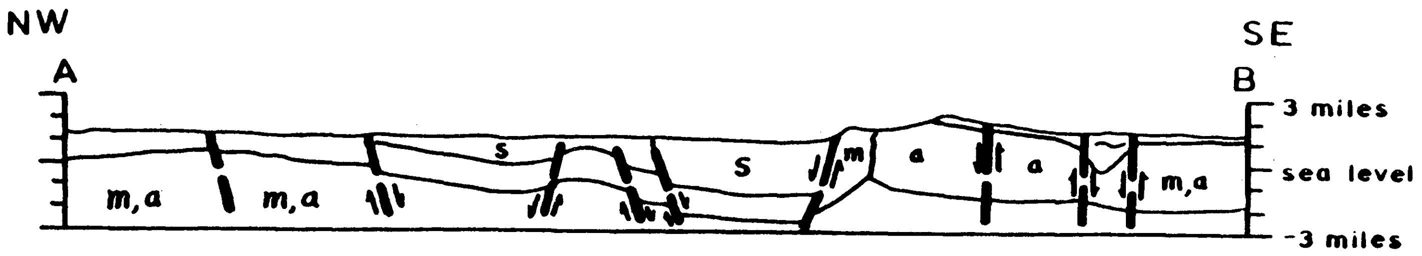


Figure 12. Structural section along the profile A-B shown in Figure 11-a. Vertical scale same as horizontal (from Woodward et al., 1975).

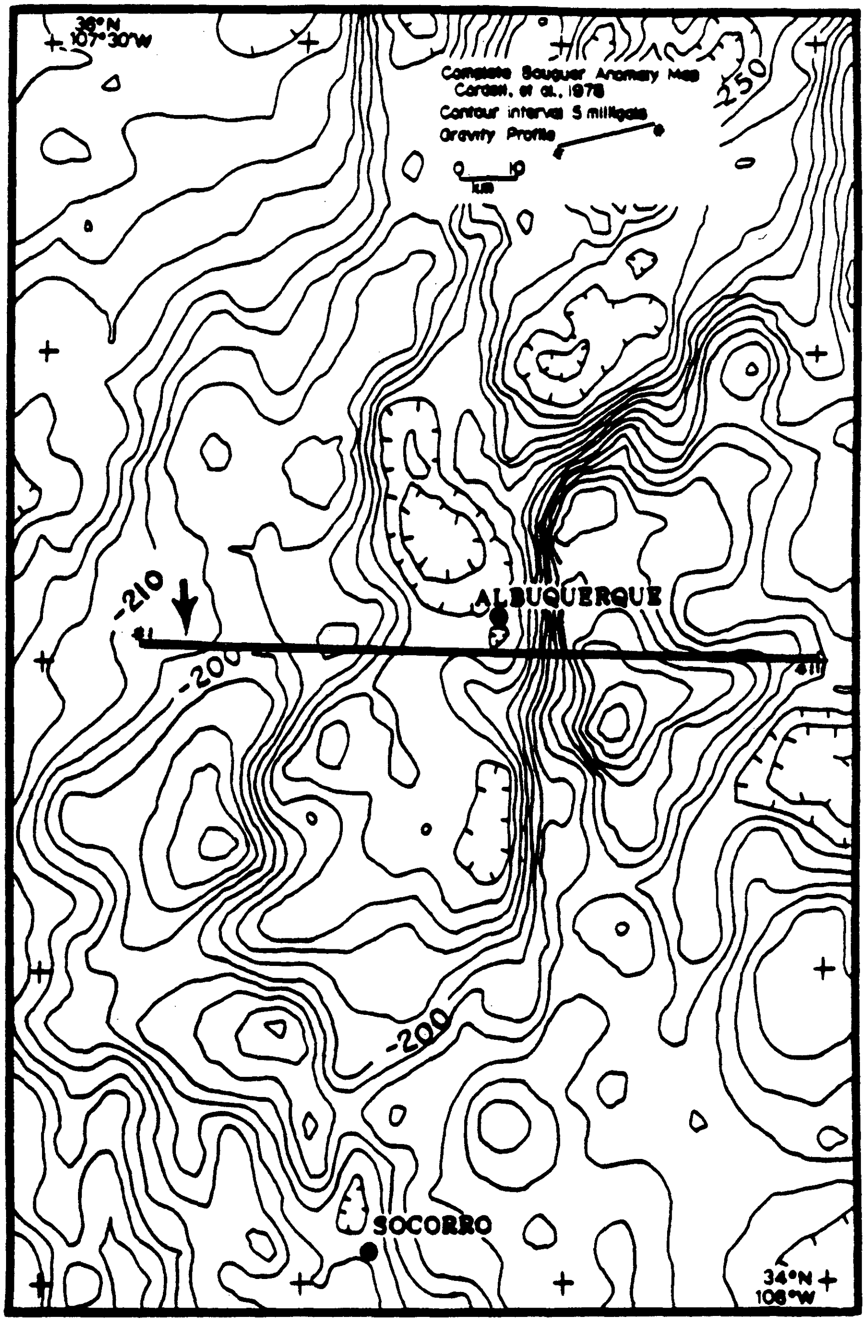


Figure 13. Complete Bouguer anomaly map of the Albuquerque basin (from Birch, 1982).

in Figure 13, at the same location as the profile #1 shown in Figure 11-a, was modeled as a multiple-faulted graben by Birch, and the model is shown in Figure 14. From Birch's study, the surficial deposits in the basin which are Neogene (i.e., Miocene to Recent) in age with a density of 2.20 g/cm^3; those at depth are Paleogene, 2.40 g/cm^3, Mesozoic, 2.57 g/cm^3, and Paleozoic, 2.67 g/cm^3. The low-density, high-porosity Neogene deposits contain large quantities of groundwater. The purpose of Birch's study was determination of the extent and thickness of Neogene deposits in the Albuquerque basin. I used Birch's gravity data and digitized the residual anomaly from Figure 14 as the observed data used in the following test. The digitized gravity anomaly is shown as dashed line in Figure 15.

Because accurate elevations were not available along the profile in Figure 14, I assumed that the observation points are all located on a datum of 2.5 kilometers above sea level, since the surface elevations are all between 1.5 and 2.5 kilometers. This had been shown to give about 1 mGal error for the response calculated on the datum using Birch's model. This error is not significant compared with the 60 mGal magnitude of the residuals. Therefore, I used Birch's model as the skeleton of the initial model, except in flattening the top surface of the model to the datum of 2.5

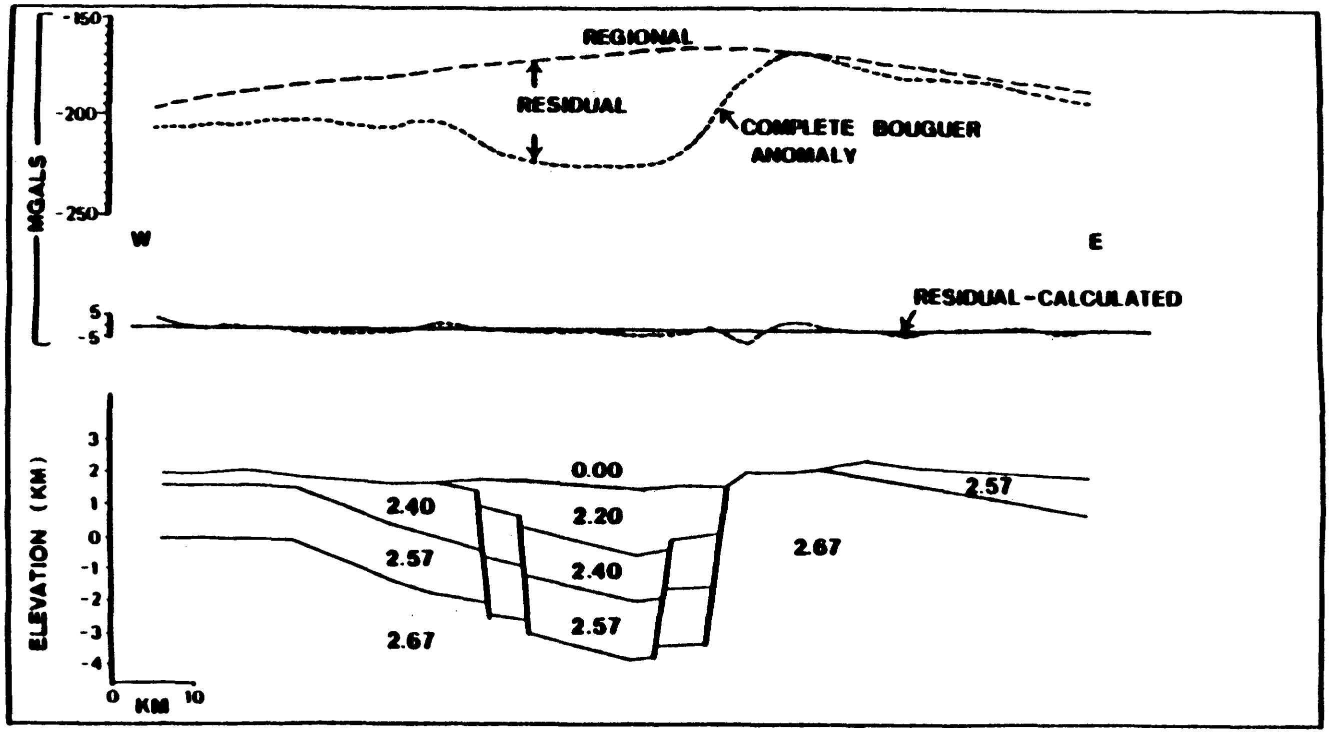


Figure 14. Gravity anomaly of the profile #1 shown in Figure 13 and the model interpreted by Birch (from Birch, 1982).

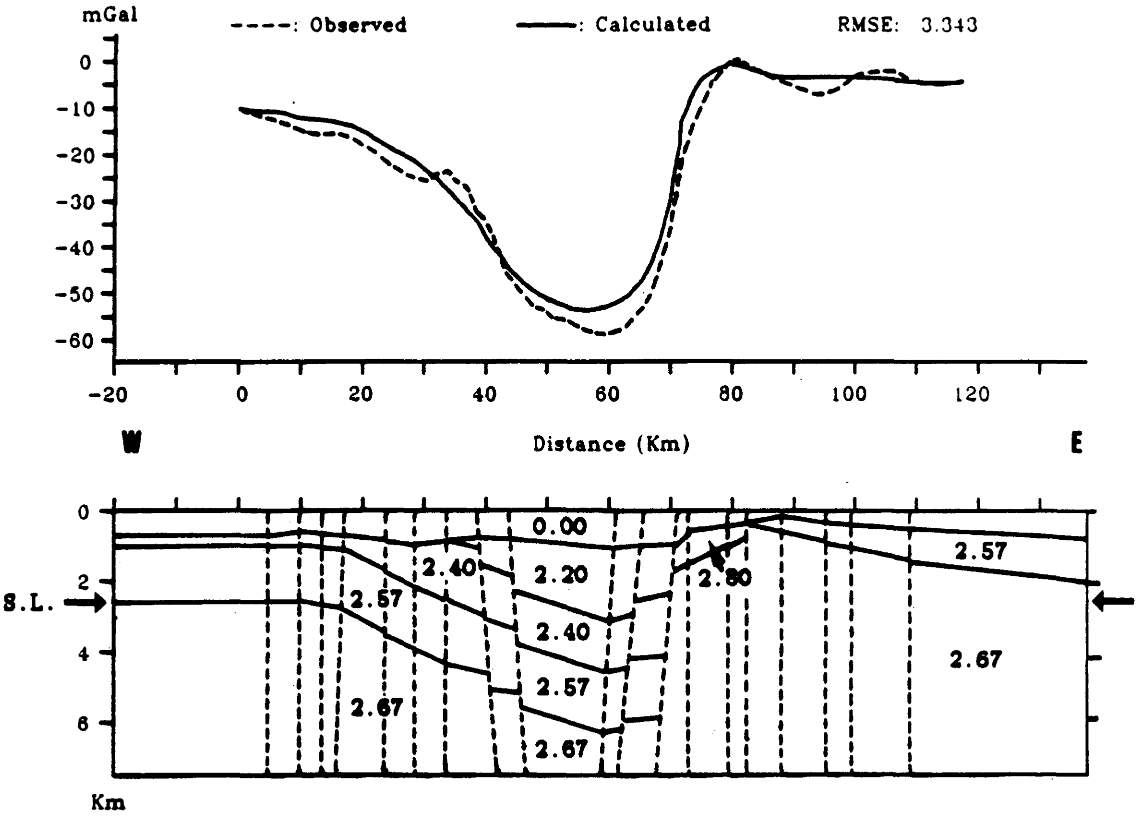


Figure 15. Residual gravity anomaly of the profile #1 and an initial model modified from Birch's model shown in Figure 14.

kilometers above sea level, and adding a 2.80 g/cm^3 high-density source body on the eastern margin of the Albuquerque basin. This initial model is shown in Figure 15. The high density body on the initial model was intended to fit the gravity high occurring on the eastern margin of the Albuquerque basin which can be clearly seen in Figures 13 and 14. This addition is consistent with the outcrops of intrusive rocks, schist, gneiss, and greenstone in Figure 11-a.

The initial model was parameterized using 20 different column blocks and 6 different layers, which include an air layer with a density of 0.00 g/cm^3. The mean density used in this model is 2.67 g/cm^3. The density contrast for the air layer is always kept zero, for which it does not give significant contribution. The initial model response is shown as the solid line in Figure 15. The RMSE between the observations and the model response is 3.343 mGal.

Several constraints were used in inverting the model. First, the topmost layer was fixed; the densities, depths, and dips of this layer were set to have zero allowable change. Second, the densities of the model were constrained to be layer-related, i.e., each layer is kept at the same density. There are six different layer densities in this model: 0.00, 2.20, 2.40, 2.57, 2.67, and 2.80 g/cm^3. Third, the tenth fault line, from the left end of the profile, is constrained to have zero throw, which is to be expected as it is a fold axis. Fourth, the depths of the third layer of the eighth and ninth blocks are constrained within a 0.5 kilometer range, since these two depths would change too much if unconstrained. Fifth, the densities of the bedrock layer are constrained to have zero allowable change range. A density of 2.67g/cm^3 was used as the mean density in the model. The RMSE limit was set to be 2 mGal.

Figure 16 shows the calculated model returned by the program. This calculated model satisfies all of the constraints mentioned above. The RMSE is 1.842 mGal. From the comparison of the initial model (Figure 15) and the calculated model (Figure 16), we find that the density of Neogene deposits, called Layer-1 hereafter, has been changed from 2.20 to 2.17 g/cm^3, and, in addition, the thicknesses on the west side have been decreased while those on the east side have been increased. For the Paleogene formation, also called Layer-2, the density has been changed from 2.40 to 2.39 g/cm^3, while the depths are only changed slightly. For the Mesozoic formation (Layer-3), the density has been changed from 2.57 to 2.56 g/cm^3, while the depths on the west side are changed significantly. In addition, some

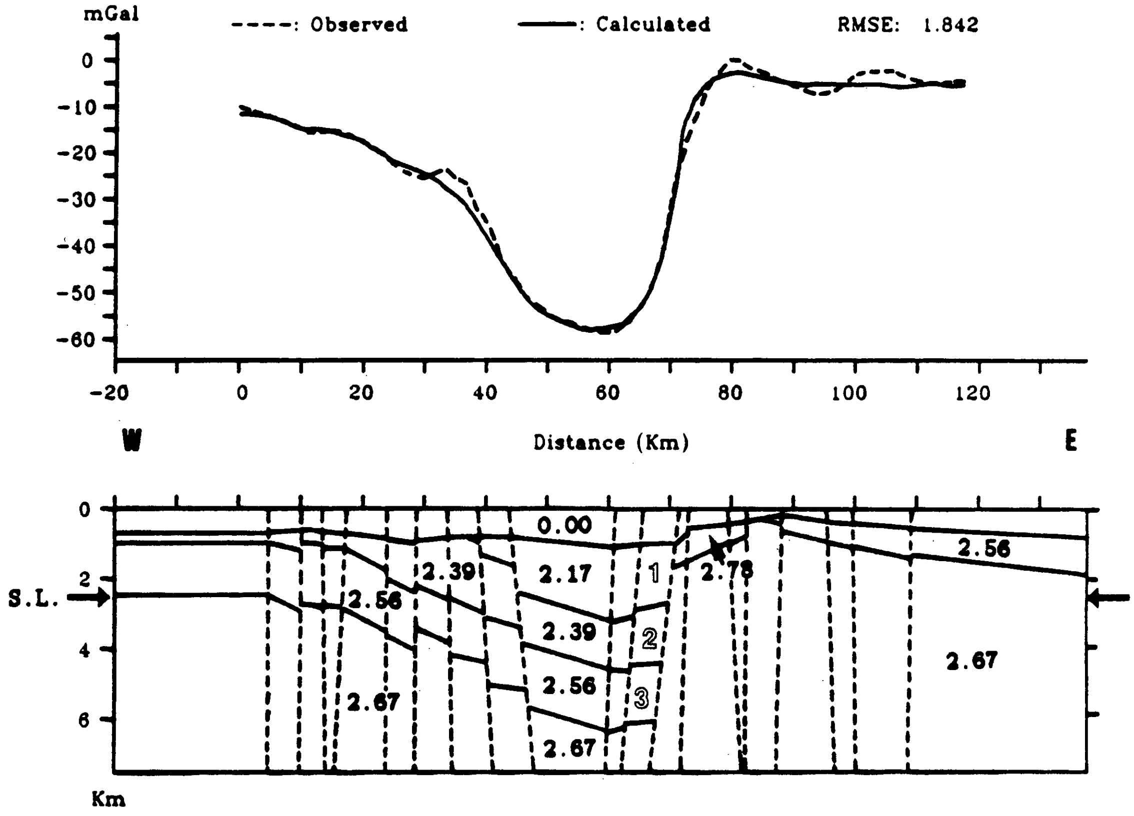


Figure 16. The calculated model with constraints based on the initial model shown in Figure 15. The numbers 1, 2 and 3 identify the layers referred to in text and Table 1.

normal faults have been interpreted under the Colorado plateau west of the Albuquerque basin. In contrast, the depths on the east side are not changed much. The density of the intrusive rock has been changed from 2.80 to 2.78 g/cm^3, while the thicknesses are not changed much. The densities of the topmost layer and the bottom layer were kept unchanged as required. The fault lines on the west side have not been changed much, while those on the east side have been tilted significantly to the east.

The density comparison among the average densities of the well logs mentioned in the Birch's paper, Birch's model, and the resulting model calculated by the program are shown in Table 1. The well densities for the three major strata are 2.18, 2.41, and 2.54 g/cm^3, respectively. Birch's densities are 2.20, 2.40, and 2.57 g/cm^3, respectively. The densities calculated by program are 2.17, 2.39, and 2.56 g/cm^3, respectively. The greatest difference between the Birch's densities and well densities is 0.03 g/cm^3, while that between the well densities and the calculated densities is 0.02 g/cm^3. This result shows that the rule-based inverse modeling program can implement geological constraints in the inversion and can provide an interpretation as good as Birch's. Furthermore, it can save a lot of time compared to the conventional trial and error method.

Table 1. Comparison of the densities of well logs, Birch's results, and this study. (unit: g/cm^3)

	Well logs	Birch's results	This study
Layer 1	2.18	2.20	2.17
Layer 2	2.41	2.40	2.39
Layer 3	2.54	2.57	2.56

CONCLUSIONS AND RECOMMENDATIONS

The rule-based gravity inversion modeling program has been tested with satisfactory results using both synthetic and real gravity data to interpret subsurface structures, in which a variety of geological constraints were incorporated. These constraints include allowable correction ranges for the model parameters (bed densities and depths and dips, and fault locations and dips); the continuity of layer densities; the continuity of layer boundaries; constant fault throws; and fixing of some model parameters. From the results of testing, it can be concluded that the rule base not only can embed the above constraints in gravity inversion and check if the calculations satisfy them, but it also plays an important role in modification of intermediate models by reweighting parameters. The resulting program can ensure that a geologically reasonable result is obtained, and it also can be used as a gravity modeling tool to replace the conventional trial and error method.

The program developed in this research can be improved further. For example, the rule-based system which performs the modification of unsatisfactory results returned from the inversion can be refined in its handling of unconstrained parameters. The unconstrained parameters are globally confined to be within the default ranges, such as 0 - 180 degrees for the fault dip angles, and 0 - 3.5 g/cm^3 for densities. These parameters are only changed slightly by a random process to avoid the inversion calculation being occasionally trapped in a loop without any improvement on the current model. This part of the rule base is still in a rudimentary stage. It would be worthwhile to create more rules for the modification of the unconstrained parameters.

In addition, further study of the adjustment of the weighting factors is necessary. The weights are currently adjusted only by a few constant values preset by the rule base, such as multiplication by 2 or division by 3; this might not be suitable in general. Furthermore, the present method used to adjust the weighting factors only affects a single parameter independently. However, there are relationships among the adjustments for different parameters. Theoretically, reducing a weighting factor should be equivalent to increasing other weights instead. More knowledge about the adjustments between the different weighting factors can certainly help the inversion to be more efficient in finding a satisfactory model.

Another worthy task would be to reduce the computations in establishing the matrix A , since each element of the matrix requires lot of computation. In this computation, the geological parameters of the model must first be transformed into vertex coordinates, then the results used to calculate the model response. Whenever the parameters are updated or perturbed, they need to be transformed to vertex coordinates. This has been proved time-consuming. To reduce the computation time, the calculation for establishing matrix A should be improved, although the program already works much faster than using the trial and error method.

Both goals of the research have been attained: the FORTRAN inversion routines have been revised to interact with the CLIPS expert system shell; and a rule-based gravity interpretation program has been created to incorporate geological constraints into the inversion process. The achievements of this research will make it easier to construct a higher-level gravity interpretation expert system which will more closely resemble the actions of a human interpreter.

REFERENCES

Birch, F.S., 1982, Gravity models of the Albuquerque basin, Rio Grande rift, New Mexico: Geophysics, 47, 1185-1197.

Brooke, T.M., 1988, Software review of COSMIC CLIPS: AI Expert, April, 71-73.

Corbato, C.E., 1965, A least-squares procedure for gravity interpretation: Geophysics, 30, 228-233.

COSMIC, 1987, CLIPS reference manual, version 4.01: NASA, Johnson Space Center, Houston, Texas.

Center for Potential Fields Studies (CPFS), 1985, Two-dimensional gravity inversion program, GRV2D: CPFS, Colorado School of Mines, Golden, Colorado.

Hayes-Roth, F., Waterman, D.A., Lenat, D.B., 1983, Building expert systems: Addison-Wesley Publishing Company, Reading, Massachusetts, 444 p.

Hoerl, A.E., Kennard, R.W., 1970, Ridge regression: biased estimation for nonorthogonal problems: Technometrics, v. 12, n. 1, 55-67.

Inman, J.R., 1975, Resistivity inversion with ridge regression: Geophysics, 40, 798-817.

Kelley, V.C., 1977, Geology of Albuquerque basin, New Mexico: Memoir 33, New Mexico Bureau of Mines and Mineral Resources, Socorro, 60 p.

Marquardt, D.W., 1963, An algorithm for least-squares estimation of nonlinear parameters: J. Soc. Indust. Appl. Math., v. 2, v. 2, 431-441.

Marquardt, D.W., 1970, Generalized inverses, ridge regression, biased linear estimation, and nonlinear estimation: Technometrics, Vol. 12, No. 3, 591-612.

Roy, A., 1962, Ambiguity in geophysical interpretation: Geophysics, 27, 90-99.

Stanford, 1987, VPLOT graphics package: Stanford University, Palo Alto, California.

Talwani, M., Worzel, J.L., Landisman, M., 1959, Rapid gravity computations for two-dimensional bodies with application to the Mendocino submarine fracture zone: J. Geophys. Res., 30, 49-59.

Waterman, D.A., 1986. A guide to expert systems: Addison-Wesley Publishing Company,Reading, Massachusetts, 419 p.

Woodward, L.A., Callender, J.F., Zilinski, R.E., 1975, Tectonic map of the Rio Grande rift, New Mexico: The Geological Society of America, Boulder, Colorado.

APPENDIX A

Listing of FORTRAN Subroutines

```
c--------------------------------------------------------------------------c
c     This routine is the start routine called from clips to do the        c
c     2-D gravity inversion using Marquardt iteration method               c
c     Arguments: body    initial model filename                            c
c                obs     observed data filename                            c
c                cbod    calculated model filename                         c
c                cdata   calculated data filename                          c
c                cpar    controlled parameters and weighting factors       c
c                        data filename                                     c
c--------------------------------------------------------------------------c
      subroutine inv2d(body,obs,cbod,cdata,cpar)
      integer body,obs,cbod,cdata,cpar
      character*11 bodfil,cbodf,obsfil,calfil,parfil
      dimension doc(300)
      common/gpars/fx(300),fz(300),grav(300),ograv(300),npts,gfac
      common/files/bodfil,cbodf,obsfil,calfil,parfil
      common/geov1/xf(25),dipf(25),zb(25,8),dipb(25,8),rhob(25,8)
      common/ibod/xfib(25),dfib(25),zbib(25,8),dbib(25,8),rbib(25,8)
      common/geov2/td,xlen,ext,z0,nf,xb(25),lay(25)
      common/box/xbl,zbl,dbl,xbt,zbt,dbt,xbr,zbr,dbr,xbd,zbd,dbd
      common/vert/x(10),z(10),nv
      common/mden/den
      data gfac/6.67/
      call loadc(bodfil,body)
      open(11,file=bodfil,blank='null')
      read(11,*) td,xlen,ext,z0,nf,den
      read(11,*) (xfib(i),dfib(i),i=1,nf)
      nf1=nf+1
      do 1 i=1,nf1
      read(11,*) xbo,lyo
      do 2 j=1,lyo
      read(11,*) zbib(i,j),dbib(i,j),rbib(i,j)
      rbib(i,j)=rbib(i,j)-den
    2 continue
    1 continue
      call loadc(obsfil,obs)
      open(12, file=obsfil)
      read(12,*) npts
      read(12,*) (fx(i),fz(i),ograv(i),i=1,npts)
      close(11)
      close(12)
      call loadc(cbodf,cbod)
      if((cbodf.eq.'nmod').or.(cbodf.eq.'NMOD')) goto 33
      open(13,file=bodfil)
      goto 34
   33 open(13,file=cbodf)
   34 read(13,*) td,xlen,ext,z0,nf,den
      read(13,*) (xf(i),dipf(i),i=1,nf)
      do 4 i=1,nf1
      read(13,*) xb(i),lay(i)
      do 5 j=1,lay(i)
      read(13,*) zb(i,j),dipb(i,j),rhob(i,j)
      rhob(i,j)=rhob(i,j)-den
```

```
    5 continue
    4 continue
      close(13)
      call bline
      do 3 i=1,npts
      grav(i)=blocks(i)
      doc(i)=ograv(i)-grav(i)
    3 continue
      rmsi=rms(doc,npts)
      call loadc(calfil,cdata)
      open(14,file=calfil)
      do 6 i=1,npts
    6 write(14,*) fx(i),grav(i)
      close(14)
      call loadc(parfil,cpar)
      call graph(rmsi)
      call stikp
      return
      end
c---------------------------------------------------------------------------c
c   This routine creates the border lines of the model without any          c
c   arguments                                                               c
c---------------------------------------------------------------------------c

      subroutine bline
      common/box/xbl,zbl,dbl,xbt,zbt,dbt,xbr,zbr,dbr,xbd,zbd,dbd
      common/geov2/td,xlen,ext,z0,nf,xb(25),lay(25)
      xbl=-ext
      zbl=z0
      dbl=90.
      xbt=0.
      zbt=z0
      dbt=0.
      xbr=xlen+ext
      zbr=z0
      dbr=90.
      xbd=0.
      zbd=z0+td
      dbd=0.
      return
      end

c---------------------------------------------------------------------------c
c   This function accumulates the response of the model at the observed     c
c   point nm                                                                c
c   Arguments:  nm    observation point number                              c
c---------------------------------------------------------------------------c

      function blocks(nm)
      common/geov1/xf(25),dipf(25),zb(25,8),dipb(25,8),rhob(25,8)
      common/geov2/td,xlen,ext,z0,nf,xb(25),lay(25)
      common/box/xbl,zbl,dbl,xbt,zbt,dbt,xbr,zbr,dbr,xbd,zbd,dbd
      grav1=block1(nm,1,xbl,zbl,dbl,xf(1),z0,dipf(1))
```

```
      do 1 i=2,nf
    1 grav1=grav1+block1(nm,i,xf(i-1),z0,dipf(i-1),xf(i),z0,dipf(i))
      blocks=grav1+block1(nm,nf+1,xf(nf),z0,dipf(nf),xbr,zbr,dbr)
      return
      end

c-----------------------------------------------------------------------c
c   This function calculates response of block nb at the observed point c
c   nm and returns the values to the function blocks                    c
c   Arguments:  nm    observation point number                          c
c               nb    block number                                      c
c               xl,zl,dl    x-coordinate, z-coordinate and dip-angle    c
c                           of the left bound line of block nb          c
c               xr,zr,dr    x-coordinate, z-coordinate and dip-angle    c
c                           of the right bound line of block nb         c
c-----------------------------------------------------------------------c

      function block1(nm,nb,xl,zl,dl,xr,zr,dr)
      common/gpars/fx(300),fz(300),grav(300),ograv(300),npts,gfac
      common/geov1/xf(25),dipf(25),zb(25,8),dipb(25,8),rhob(25,8)
      common/geov2/td,xlen,ext,z0,nf,xb(25),lay(25)
      common/box/xbl,zbl,dbl,xbt,zbt,dbt,xbr,zbr,dbr,xbd,zbd,dbd
      common/boxi/xbi(10),zbi(10),nvi
      common/vert/x(10),z(10),nv
      dimension xi(10),zi(10)
      l=lay(nb)
      rhob(nb,l+1)=0.
      gravb=0.
c      write(*,*) nm,nb,xl,zl,dl,xr,zr,dr
 1    call xpoint(xl,zl,dl,xbt,zbt,dbt,xp1,zp1)
      xi(1)=xp1
      zi(1)=zp1
      call xpoint(xbt,zbt,dbt,xr,zr,dr,xp2,zp2)
      xi(2)=xp2
      zi(2)=zp2
      call xpoint(xr,zr,dr,xb(nb),zb(nb,l),dipb(nb,l),xp3,zp3)
      xi(3)=xp3
      zi(3)=zp3
      call xpoint(xb(nb),zb(nb,l),dipb(nb,l),xl,zl,dl,xp4,zp4)
      xi(4)=xp4
      zi(4)=zp4
c      print *, 'from xpoint output'
c      write(*,*) (xi(i2),zi(i2),i2=1,4)
      if(l.ne.lay(nb)) goto 2
      call vertex(xi,zi,xl,zl,dl,xr,zr,dr)
c      print *, 'from vertex output'
c      write(*,*) (x(ii),z(ii),ii=1,nv)
      goto 11
  2   call vertex1(xi,zi,xb(nb),zb(nb,l),dipb(nb,l))
c      print *, 'from vertex1 output'
c      write(*,*) (x(i1),z(i1),i1=1,nv)
 11   rho=rhob(nb,l)-rhob(nb,l+1)
      gravb=gravb+corbat(x,z,nv,fx(nm),fz(nm),rho)
```

```
      l=l-1
      if(l.ge.1) goto 1
      blockl=gravb
      return
      end

c--------------------------------------------------------------------c
c  This subroutine returns the intersection point of two lines      c
c  Arguments:  x1,z1,d1     x-coordinate, z-coordinate, dip-angle    c
c                           of the first line                        c
c              x2,z2,d2     x-coordinate,z-coordinate, dip-angle     c
c                           of the second line                       c
c              xp,zp        x-coordinate, z-coordinate of the        c
c                           intersection                             c
c--------------------------------------------------------------------c
      subroutine xpoint(x1,z1,d1,x2,z2,d2,xp,zp)
      d11=d1*0.0174532
      d22=d2*0.0174532
      d1m=mod(d1,180.)
      d2m=mod(d2,180.)
      if(d1m.eq.d2m) goto 99
      if((d1.eq.0.).and.(d2.eq.90.)) then
      xp=x2
      zp=z1
      goto 100
      endif
      if(d1.eq.0.) then
      zp=z1
      xp=x2+(zp-z2)/tan(d22)
      goto 100
      endif
      if((d1.eq.90.).and.(d2.eq.0.)) then
      xp=x1
      zp=z2
      goto 100
      endif
      if(d1.eq.90.) then
      xp=x1
      zp=z2+(xp-x2)*tan(d22)
      goto 100
      endif
      if(d2.eq.0.) then
      zp=z2
      xp=x1+(zp-z1)/tan(d11)
      goto 100
      endif
      if(d2.eq.90.) then
      xp=x2
      zp=z1+(xp-x1)*tan(d11)
      goto 100
      endif
      xp=(z2-z1-x2*tan(d22)+x1*tan(d11))/(tan(d11)-tan(d22))
      zp=z2+(xp-x2)*tan(d22)
```

```
      goto 100
   99 print *, 'wrong in calling xpoint'
      stop
  100 return
      end

c-----------------------------------------------------------------------c
c    This routine checks the vertices of the block ridge to avoid        c
c    using wrong vertex values                                           c
c    Arguments:  xin,zin    array of vertice coordinates to be checked   c
c                xl,zl,dl   x-coordinate, z-coordinate, and dip-angle    c
c                           of the left bound line                       c
c                xr,zr,dr   x-coordinate, z-coordinate, and dip-angle    c
c                           of the right bound line                      c
c    Variables:  common boxi    store the bound vertice of the ith block c
c                common vert    store the vertice of the one source body c
c-----------------------------------------------------------------------c
      subroutine vertex(xin,zin,xl,zl,dl,xr,zr,dr)
      common/boxi/xbi(10),zbi(10),nvi
      common/vert/x(10),z(10),nv
      dimension xin(10),zin(10)
      if(xin(4).lt.xin(3)) goto 2
      do 1 i=1,2
      xbi(i)=xin(i)
    1 zbi(i)=zin(i)
      call xpoint(xl,zl,dl,xr,zr,dr,xp,zp)
      xbi(3)=xp
      zbi(3)=zp
      nv=3
      goto 99
    2 do 3 i=1,4
      xbi(i)=xin(i)
    3 zbi(i)=zin(i)
      nv=4
   99 do 4 i=1,nv
      x(i)=xbi(i)
    4 z(i)=zbi(i)
      return
      end

c-------------------------------------------------------------------------c
c  This routine checks the inner layers' vertices avoid being used wrongly c
c
c-------------------------------------------------------------------------c
      subroutine vertex1(xin,zin,xbl,zbl,dbl)
      common/box/xbl,zbl,dbl,xbt,zbt,dbt,xbr,zbr,dbr,xbd,zbd,dbd
      common/boxi/xbi(10),zbi(10),nvi
      common/vert/x(10),z(10),nv
      dimension xin(10),zin(10)
      if(nvi.eq.3) goto 9
      if(zin(4).le.zbi(4)) goto 2
      do 1 i=1,3
      x(i)=xin(i)
```

```
 1 z(i)=zin(i)
   call xpoint(xbl,zbl,dbl,xbd,zbd,dbd,xp,zp)
   x(4)=xp
   z(4)=zp
   x(5)=xbi(4)
   z(5)=zbi(4)
   nv=5
   goto 99
 2 if(zin(4).ge.zbi(1)) goto 4
   do 3 i=2,3
   x(i)=xin(i)
 3 z(i)=zin(i)
   call xpoint(xbl,zbl,dbl,xbt,zbt,dbt,xp,zp)
   x(1)=xp
   z(1)=zp
   nv=3
   goto 99
 4 if(zin(3).le.zbi(3)) goto 6
   do 5 i=1,2
   x(i)=xin(i)
 5 z(i)=zin(i)
   x(3)=xbi(3)
   z(3)=zbi(3)
   call xpoint(xbl,zbl,dbl,xbd,zbd,dbd,xp,zp)
   x(4)=xp
   z(4)=zp
   x(5)=xin(4)
   z(5)=zin(4)
   nv=5
   goto 99
 6 if(zin(3).ge.zbi(2)) goto 7
   x(1)=xin(1)
   z(1)=zin(1)
   call xpoint(xbl,zbl,dbl,xbt,zbt,dbt,xp,zp)
   x(2)=xp
   z(2)=zp
   x(3)=xin(4)
   z(3)=zin(4)
   nv=3
   goto 99
 7 do 8 i=1,4
   x(i)=xin(i)
 8 z(i)=zin(i)
   nv=4
   goto 99
 9 if((zin(4).ge.zbi(1)).and.(zin(4).le.zbi(3))) goto 11
   do 10 i=2,3
   x(i)=xin(i)
10 z(i)=zin(i)
   call xpoint(xbl,zbl,dbl,xbt,zbt,dbt,xp,zp)
   x(1)=xp
   z(1)=zp
   nv=3
```

```
      goto 99
   11 if((zin(3).ge.zbi(2)).and.(zin(3).le.zbi(3))) goto 12
      x(1)=xin(1)
      z(1)=zin(1)
      call xpoint(xbl,zbl,dbl,xbt,zbt,dbt,xp,zp)
      x(2)=xp
      z(2)=zp
      x(3)=xin(4)
      z(3)=zin(4)
      nv=3
   12 do 13 i=1,3
      x(i)=xin(i)
   13 z(i)=zin(i)
      nv=3
   99 return
      end

c---------------------------------------------------------------------c
c  This subroutine links the forward model to the marquardt inversion  c
c  algorithm. values defining the initial guess body and the model     c
c  output for the initial guess are loaded into arrays (p and m)       c
c  compatible with the inversion routine. also, the values of the      c
c  observed field to be fitted are loaded in o and flags indicating    c
c  fixed parameters are loaded in fixp.                                c
c---------------------------------------------------------------------c
      subroutine stikp
      real m
      integer fixp
      character*11 bodfil,cbodf,obsfil,calfil,parfil
c      dimension ifix(300)
      common/blck11/m(300),im
      common/blck12/o(300),io
      common/blck14/p(300),ip
      common/blck32/fixp(300),if
      common/gpars/fx(300),fz(300),grav(300),ograv(300),npts,gfac
      common/files/bodfil,cbodf,obsfil,calfil,parfil
      common/geov1/xf(25),dipf(25),zb(25,8),dipb(25,8),rhob(25,8)
      common/geov2/td,xlen,ext,z0,nf,xb(25),lay(25)
      common/box/xbl,zbl,dbl,xbt,zbt,dbt,xbr,zbr,dbr,xbd,zbd,dbd
      common/boxi/xbi(10),zbi(10),nvi
      common/vert/x(10),z(10),nv
c      data ifix,fixp/300*0,300*0/
      if(bodfil(1:5).eq.'     ') then
      write(*,*) 'no initial guess body'
      return
      endif
      if(cbodf(1:5).eq.'     ') then
      write(*,*) 'no calculated/modified model '
      return
      endif
      if(obsfil(1:5).eq.'     ') then
      write(*,*) 'no observed gravity'
      return
```

```
      endif
c---load parameter matrix noting fixed elements
c      write(*,*) 'initial body parameters for inversion:'
c      write(*,*) 'fault number: ',nf
      do 1 i=1,nf
      p(2*i-1)=xf(i)
      p(2*i)=dipf(i)
c    1 write(*,666) i,p(2*i-1),z0,p(2*i)
    1 continue
      nf1=2*nf
      do 2 i2=1,nf+1
c      write(*,*) 'block #',i2
      do 3 i3=1,lay(i2)
      p(3*i3-2+nf1)=zb(i2,i3)
      p(3*i3-1+nf1)=dipb(i2,i3)
      p(3*i3+nf1)=rhob(i2,i3)
c    3 write(*,667) i3,xb(i2),zb(i2,i3),dipb(i2,i3),rhob(i2,i3)
    3 continue
      nf1=nf1+3*lay(i2)
    2 continue
      ip=nf1
  666 format(1x,' fault #',i2,5x,'x =',f7.2,'   z =',f7.2,'  dip =',
     1 f7.2)
  667 format(1x,'lay #',i2,5x,'x =',f7.2,'   z =',f7.2,'  dip =',f7.2,
     1 '  rho =',f7.2)
      npfix=2*nf
      do 151 i=1,nf+1
      fixp(3*lay(i)-2+npfix)=1
      fixp(3*lay(i)-1+npfix)=1
  151 npfix=npfix+3*lay(i)
      if=ip
c
c---load model data
      do 200,i=1,npts
      m(i)=grav(i)
  200 continue
      im=npts
c
c---load observed data
      do 300,i=1,npts
      o(i)=ograv(i)
  300 continue
      io=npts
c
c---commence inverting
      call invers
c      write(*,*)'inversion complete.'
      return
      end
c-----------------------------------------------------------------------------------c
c   Corbat calculates gravitational effect due to body at field point x,z        c
c                                                                                   c
c     Arguments:                                                                    c
```

```
c               bx      array containing x coordinates of body          c
c               bz      array containing z coordinates of body          c
c               nv      number of vertices defining body                c
c               drho    density contrast for body                       c
c               x       x coordinate of field point                     c
c               z       z coordinate of field point                     c
c                                                                       c
c     Note: all distance units are in kilometers                        c
c-----------------------------------------------------------------------c
      function corbat(bx,bz,nv,x,z,drho)
      dimension bx(1), bz(1)
      common/gpars/fx(300),fz(300),grav(300),ograv(300),npts,gfac

c
      data small /1.e-5/
c
      zg = 0.0
      x1 = bx(nv) - x
      z1 = bz(nv) - z
      do 10 i = 1, nv
      x2 = bx(i) - x
      z2 = bz(i) - z
      r1 = x1*x1 + z1*z1
      r2 = x2*x2 + z2*z2
      if ( abs(r1) .lt. small .or. abs(r2) .lt. small) go to 5
      a = x2-x1
      b = z2-z1
      t = x1*x2 + z1*z2
      u = a*a + b*b
      s = x1*z2 - x2*z1
      v = .5 * alog(r2/r1)
      w = atan2(s,t)
      if(u.eq.0.) goto 5
      zg = zg + s/u * (b*v - a*w)
 5    x1 = x2
      z1 = z2
 10   continue
      corbat = 2.*gfac*drho*zg
      return
      end
c
c-------------------------------------------------------------------------c
c   Function splitp connects inversion routines to the forward model.     c
c   Inversion uses body parameters arranged in a single one dimensional array. c
c   Forward model uses the parameters arranged in two one dimensional arrays,  c
c   one for the x coordinates of body vertices, one for the z coordinates     c
c   of the vertices.  This subroutine essentially performs the inverse       c
c   transformation done in stikp.                                          c
c                                                                         c
c   Arguments:                                                            c
c             n     option for array p (n=1) or pnew (n=2)                c
c             nm    observation point number                              c
c             delp  increment for calculating first derivative            c
```

```
c              np    number of parameter to be added delp with             c
c--------------------------------------------------------------------------c
      function splitp(n,nm,delp,np)
      common/gpars/fx(300),fz(300),grav(300),ograv(300),npts,gfac
      common/geov1/xf(25),dipf(25),zb(25,8),dipb(25,8),rhob(25,8)
      common/blck14/p(300),ip
      common/blck15/dp(300),pnew(300),mnew(300),dnew(300)
      common/geov2/td,xlen,ext,z0,nf,xb(25),lay(25)
      common/box/xbl,zbl,dbl,xbt,zbt,dbt,xbr,zbr,dbr,xbd,zbd,dbd
      common/boxi/xbi(10),zbi(10),nvi
      common/vert/x(10),z(10),nv
      if(n.eq.2) goto 9
      p(np)=p(np)+delp
      do 1 i=1,nf
      xf(i)=p(2*i-1)
    1 dipf(i)=p(2*i)
      nf1=2*nf
      do 2 i2=1,nf+1
      do 3 i3=1,lay(i2)
      zb(i2,i3)=p(3*i3-2+nf1)
      dipb(i2,i3)=p(3*i3-1+nf1)
    3 rhob(i2,i3)=p(3*i3+nf1)
      nf1=nf1+3*lay(i2)
    2 continue
      p(np)=p(np)-delp
      goto 99
    9 do 4 i=1,nf
      xf(i)=pnew(2*i-1)
    4 dipf(i)=pnew(2*i)
      nf1=2*nf
      do 5 i2=1,nf+1
      do 6 i3=1,lay(i2)
      zb(i2,i3)=pnew(3*i3-2+nf1)
      dipb(i2,i3)=pnew(3*i3-1+nf1)
    6 rhob(i2,i3)=pnew(3*i3+nf1)
      nf1=nf1+3*lay(i2)
    5 continue
   99 splitp=blocks(nm)
      return
      end
c
c--------------------------------------------------------------------------c
c  Subroutine invers is a calling routine for the inversion.              c
c  common blocks for standard marquardt inversion:                        c
c                                                                         c
c          a    matrix of partial derivatives (m x n)                     c
c          at   transpose of a (n x m)                                    c
c          ata  a multiplied times at (n x n)                             c
c          p    model parameters, i.e. vertice coordinates (n x 1)        c
c          m    gravity values at field points calculated for body        c
c               defined by p matrix;  this matrix is updated each         c
c               time the inversion calculates new values for p (m x 1)    c
c          o    observed gravity values at field points;  this is         c
```

```
c                the profile the inversion attempts to match (m x 1)         c
c           d    difference between observed gravity and present             c
c                model gravity; used to determine rms error (m x 1)          c
c          geov1 changable model parameters                                  c
c          ibod  initial unchangable model parameters                        c
c   Note: matrices p,m,o and d (as well as several other matrices            c
c         utilized in subroutines) are one dimensional but are dimensioned   c
c         with a second subscript (always equal to 1) so that matrix         c
c         manipulations written for general double subscripted arrays        c
c         can be used.                                                       c
c----------------------------------------------------------------------------c
      subroutine invers
      real lam0,m,mnew
      integer fixp
      character*11 bodfil,cbodf,obsfil,calfil,parfil
      character*10 xfx,fd,bz,bd,br,xfc,dpfc,zbc,dpbc,robc,wtv
      character*48 fx1,fd1,bz1,bd1,br1,wtc,srms
      character*3 fi,bi2,bi3
      character*5 cxf,cdf,czb,cdb,crb
      character wtn*5, crms*7
      integer ifx(12),ifd(12),ibz(12),ibd(12),ibr(12),iwt(12),irs(12)
      common/rmse/crms
      common/gpars/fx(300),fz(300),grav(300),ograv(300),npts,gfac
      common/files/bodfil,cbodf,obsfil,calfil,parfil
      common/pscale/delp(5),afa1(5),npf,afa(300)
      common/blck1/a(300,300),ia,ja
      common/blck2/at(300,300),iat,jat
      common/blck3/ata(300,300),iata,jata
      common/blck11/m(300),im
      common/blck12/o(300),io
      common/blck13/d(300),id
      common/blck14/p(300),ip
      common/blck15/dp(300),pnew(300),mnew(300),dnew(300)
      common/blck28/t2(300)
      common/blck29/atd(300)
      common/blck31/aug(300,600),iaug,jaug
      common/blck32/fixp(300),if
      common/geov1/xf(25),dipf(25),zb(25,8),dipb(25,8),rhob(25,8)
      common/geov2/td,xlen,ext,z0,nf,xb(25),lay(25)
      common/ibod/xfib(25),dfib(25),zbib(25,8),dbib(25,8),rbib(25,8)
      common/mden/den
c---set up exit criteria
      iter=0
      itlim=20
      open(10,file=parfil)
      read(10,*) itm
      if(itm.gt.1) goto 66
      read(10,*)rmslim
c---calculate d matrix and initial error
  66  do 4 i=1,im
   4  d(i)=o(i)-m(i)
      id=im
      rmsold=rms(d,id)
```

```
c---run information
      open(7,file='invrun.dat')
      rewind(7)
      write(7,*)'General marquardt inversion'
      write(7,999)
      call printm(12,7)
      write(7,*)io,' observations'
      write(7,1000)
      call printm(11,7)
      write(7,*)im,' model points'
      write(7,1001)
      call printm(14,7)
      write(7,*)ip,' total parameters'
      write(7,1002)rmsold,rmslim,itlim
      if(itm.gt.1) goto 90
c---load a matrix noting fixed parameters
      read(10,*) (delp(i),afal(i),i=1,5)
      do 5 i=1,npf-1,2
      afa(i)=afal(1)
    5 afa(i+1)=afal(2)
      do 6 i=npf+1,ip-2,3
      afa(i)=afal(3)
      afa(i+1)=afal(4)
    6 afa(i+2)=afal(5)
   90 read(10,*,end=91) mxn
      fixp(mxn)=1
      goto 90
   91 close(10)
      npf=2*nf
      nf1=npf
      write(*,1003)rmsold
  110 iter=iter+1
      write(7,1004)iter
      do 20 i=1,im
   20 t2(i)=splitp(1,i,0.0,1)
      do 130,j=1,ip
      if (fixp(j).eq.1) then
        do 120,k=1,im
        a(k,j)=0.0
  120   continue
        go to 130
      else
      if(j.gt.npf) goto 71
      mj=mod(j,2)
      if(mj.eq.1) call deriv(im,j,delp(1),afa(j))
      if(mj.eq.0) call deriv(im,j,delp(2),afa(j))
      goto 130
   71 jm=j-npf
      jn=mod(jm,3)
      if(jn.eq.1) call deriv(im,j,delp(3),afa(j))
      if(jn.eq.2) call deriv(im,j,delp(4),afa(j))
      if(jn.eq.0) call deriv(im,j,delp(5),afa(j))
      endif
```

```
  130 continue
      ia=im
      ja=ip
c***general marguardt inversion*********************************************
c---calculate ata
      do 13 i=1,ia
      do 13 j=1,ja
   13 at(j,i)=a(i,j)
      iat=ja
      jat=ia
      do 11 i=1,ja
      do 11 j=1,ja
   11 ata(i,j)=0.
      do 12 i=1,ja
      do 22 j=1,ja
      do 32 k=1,ia
   32 ata(i,j)=ata(i,j)+at(i,k)*a(k,j)
   22 continue
    12 continue
      iata=iat
      jata=ja
c---calculate an initial value for lam0 to be used for the first iteration
      if (iter.eq.1) then
        trac=0.0
        do 320,i=1,iata
        trac=trac+ata(i,i)
  320   continue
        lam0=trac
       endif
c---find rms error using lam0 and new a matrix
      call solve(lam0)
      call addm
      do 330,i=1,im
      mnew(i)=splitp(2,i,0.0,1)
      dnew(i)=o(i)-mnew(i)
  330 continue
      rms0=rms(dnew,id)
c---lamda search
      call lamda(lam0,rms0,rmsold)
      drms=abs(rmsold-rms0)
      prms=rms0/100.
      rmsold=rms0
c---print messages
  410 write(7,1005)lam0,rms0
      call printm(14,7)
      write(*,1006)iter,rms0
c---check for completion
      if((rms0.lt.rmslim.or.iter.eq.itlim).or.(drms.le.prms)) then
      close(7)
      close(8)
      open(1,file=cbodf)
      open(2,file=calfil)
      do 8 i=1,im
```

```
8  grav(i)=m(i)
   write(2,*) (fx(i),m(i),i=1,im)
   write(1,*) td,xlen,ext,z0,nf,den
   do 1 i=1,nf
   write(xfc,'(f10.2)') xfib(i)
   write(dpfc,'(f10.2)') dfib(i)
   write(fi,'(i3)') i
   nxf=2*i-1
   ndf=nxf+1
   write(xfx,'(f10.2)') p(nxf)
   write(fd,'(f10.2)') p(ndf)
   write(1,*) p(nxf),p(ndf)
   write(cxf,'(i5)') nxf
   write(cdf,'(i5)') ndf
   fx1='result f x '//fi//xfc//xfx//cxf
   fd1='result f d '//fi//dpfc//fd//cdf
   call storec(fx1,ifx)
   call storec(fd1,ifd)
   call assert(ifx)
   call assert(ifd)
1  continue
   do 2 i2=1,nf+1
   write(bi2,'(i3)') i2
   write(1,*) xb(i2),lay(i2)
   do 3 i3=1,lay(i2)
   write(bi3,'(i3)') i3
   write(zbc,'(f10.2)') zbib(i2,i3)
   write(dpbc,'(f10.2)') dbib(i2,i3)
   write(robc,'(f10.2)') rbib(i2,i3)+den
   nzb=3*i3-2+nf1
   ndb=nzb+1
   nrb=nzb+2
   write(bz,'(f10.2)') p(nzb)
   write(bd,'(f10.2)') p(ndb)
   write(br,'(f10.2)') p(nrb)+den
   write(1,*) p(nzb),p(ndb),p(nrb)+den
   write(czb,'(i5)') nzb
   write(cdb,'(i5)') ndb
   write(crb,'(i5)') nrb
   bz1='result b z '//bi2//bi3//zbc//bz//czb
   bd1='result b d '//bi2//bi3//dpbc//bd//cdb
   br1='result b r '//bi2//bi3//robc//br//crb
   call storec(bz1,ibz)
   call storec(bd1,ibd)
   call storec(br1,ibr)
   call assert(ibz)
   call assert(ibd)
   call assert(ibr)
3  continue
   nf1=nf1+3*lay(i2)
2  continue
     close(1)
     close(2)
```

```
      do 7 i=1,ip
      write(wtv,'(f10.5)') afa(i)
      write(wtn,'(i5)') i
      wtc='weight '//wtn//wtv
      call storec(wtc,iwt)
    7 call assert(iwt)
        call graph(rms0)
        srms='calculated-rmse '//crms
        call storec(srms,irs)
        call assert(irs)
        return
       endif
      go to 110
 999  format(///,1x,'***** run information for grv2d inversion*****',
     1 /,1x,'observed gravity (matrix o):')
 1000 format(//,1x,'Initial guess gravity (matrix m)')
 1001 format(//,1x,'Initial guess body parameters (matrix p)')
 1002 format(////,1x,'Initial RMS Error:',f12.6,/,
     1 1x,'RMS Error limit:',f12.6,/,1x,'Iteration limit:',i3)
 1003 format(/////,1x,'Initial RMS Error: ',f12.6)
 1004 format(//,1x,'Iteration #',i3)
 1005 format(5x,'Final lam0:',f12.6,' RMS Error:',f12.6,/,
     1 1x,'New p matrix:')
 1006 format(1x,'Iteration #',i3,'; RMS Error:',f12.6)
      end
c
c----------------------------------------------------------------------------c
c   This routine to calculate finite derivatives.                            c
c    Arguments:                                                              c
c               im    number of field points                                 c
c               np    index of the variable in p with respect to which       c
c                     the derivative will be taken.                          c
c               delp  the finite difference used for the calculation         c
c               afa   array of weighting factors                             c
c                                                                            c
c   A derivative is calculated for each field point and it corresponds to    c
c   the change in the gravitational attraction which would occur at that     c
c   field point if body parameter np were changed by an amount delp.         c
c   Function corbat is used for the calculation of a model response for      c
c   the unperturbed parameter set and for a parameter set with parameter     c
c   np perturbed by delp. The difference between the two model response is   c
c   then divided by delp to achieve the derivative.                          c
c                                                                            c
c   Note:  The derivatives are placed in matrix a and passed to other        c
c          routines via common blck1.                                        c
c----------------------------------------------------------------------------c
      subroutine deriv(im,np,delp,afa)
      common/blck1/a(300,300),ia,ja
      common/blck14/p(300),ip
      common/blck28/t2(300)
      do 20,i=1,im
      t1=splitp(1,i,delp,np)
      a(i,np)=(t1-t2(i))/delp*afa
```

```
 20   continue
      return
      end
c
c-------------------------------------------------------------------c
c  This function calculates root mean square error associated with  c
c  vector e containing ie differences                               c
c-------------------------------------------------------------------c
      function rms(e,ie)
      dimension e(ie)
      err=0.0
      do 10,i=1,ie
      err=err+e(i)**2
 10   continue
      rms=sqrt(err/ie)
      return
      end
c
c----------------------------------------------------------------------c
c  This routine search for value of lamda (lam0) which minimizes the   c
c  variance of the solution due to data and model error.  Variance is  c
c  estimated by the the total rms error between observed and calculated c
c  values at survey points.  Consequently the search for a lam0 must   c
c  solve the inversion equation using different values of lamda.       c
c  The solutions are sets of corrections dp to the present body parameters c
c  p.  New p (desiganted pnew) are then found based on these corrections. c
c  the forward model is  called to compute an updated model response   c
c  (designated mnew) and a new rms error (rmsnew) is calculated and    c
c  compared to the previous error.  Iterations of the search continue  c
c  until rmsnew becomes larger than the previous rms error.  The optimum c
c  lam0 is chosen as a weighted average of the lamdas corresponding to the c
c  final two rms errors.                                               c
c                                                                      c
c  The rms error from the previous iteration is saved as rmsold.  The  c
c  optimum lam0 from the previous iteration is the starting point      c
c  for the lamda search in the subsequent iteration.  Under certain,   c
c  unpredictable circumstances decreasing this initial lamda causes    c
c  divergence, i.e. the rms error knows no bounds.  To avoid this      c
c  problem a backup search detects this occurence and begins to slowly c
c  increase lamda until an optimum is reached which ussually results   c
c  in a final rms error better than for the previous iteration.        c
c  (<rmsold).                                                          c
c                                                                      c
c  Note:  Subroutine lamda finds the optimum lam0, solves the inversion c
c         equation for corrections to the model and updates the p      c
c         before returning.  The updated p is passed via common blck4. c
c----------------------------------------------------------------------c
      subroutine lamda(lam0,rms0,rmsold)
      real lam0,lamnew,lamsub,lamadd,m,mnew
      common/blck11/m(300),im
      common/blck12/o(300),io
      common/blck13/d(300),id
      common/blck14/p(300),ip
```

```
      common/blck15/dp(300),pnew(300),mnew(300),dnew(300)
      write(7,999)lam0,rms0,rmsold
c---begin reducing lam0
 10   lamnew=lam0/2.0
      call solve(lamnew)
      call addm
      do 20,i=1,im
      mnew(i)=splitp(2,i,0.0,1)
      dnew(i)=o(i)-mnew(i)
 20   continue
      rmsnew=rms(dnew,id)
      write(7,1000)lamnew,rmsnew
      if (rmsnew.lt.rms0) then
        lam0=lamnew
        rms0=rmsnew
        go to 10
       endif
c---find optimum lam0 from reduction
      lamsub=rms0/rmsnew*(lam0-lamnew)
      lam0=lam0-lamsub
      call solve(lamnew)
      call addm
      do 30,i=1,im
      mnew(i)=splitp(2,i,0.0,1)
      dnew(i)=o(i)-mnew(i)
 30   continue
      rms0=rms(dnew,io)
      write(7,1001)lam0,rms0
c---increase slowly to optimum if necessary
      if (rms0.lt.rmsold) go to 210
 110  lamnew=lam0*1.3
      call solve(lamnew)
      call addm
      do 120,i=1,im
      mnew(i)=splitp(2,i,0.0,1)
      dnew(i)=o(i)-mnew(i)
 120  continue
      rmsnew=rms(dnew,io)
      write(7,1002)lamnew,rmsnew
      if (rmsnew.lt.rms0) then
        lam0=lamnew
        rms0=rmsnew
        go to 110
       endif
c---find optimum lamda from increase
      lamadd=rms0/rmsnew*(lamnew-lam0)
      lam0=lam0+lamadd
      call solve(lamnew)
      call addm
      do 130,i=1,im
      mnew(i)=splitp(2,i,0.0,1)
      dnew(i)=o(i)-mnew(i)
 130  continue
```

```
      rms0=rms(dnew,io)
      write(7,1003)lam0,rms0
c---calculate new p,m,d
 210  do 220,i=1,ip
      p(i)=pnew(i)
 220  continue
      do 230,i=1,im
      m(i)=mnew(i)
      d(i)=dnew(i)
 230  continue
      return
 999  format(5x,'in lamda, with lam0=',f12.6,',  rms0=',f12.6,
     1  ', rmsold=',f12.6)
 1000 format(1x,'(dcr) in lamda, with lamnew=',f12.6,',  rmsnew=',f12.6)
 1001 format(5x,'reduction optimum: lam0=',f12.6,',  rms0=',f12.6)
 1002 format(1x,'(icr) in lamda, with lam0=',f12.6,',  rms0=',f12.6)
 1003 format(5x,'increase optimum: lam0=',f12.6,',  rms0=',f12.6)
      end
c
c----------------------------------------------------------------------c
c   This subroutine performs the calculation of the corrections dp to  c
c   the current model according to the general marquardt inversion     c
c   equation:                                                          c
c                                                                      c
c                                                                      c
c                  [(at)(a) + (lam)(id)](dp) = (at)(d)                 c
c                                                                      c
c   where dp are the predicted corrections to the body parameters.    c
c----------------------------------------------------------------------c
      subroutine solve(lam0)
      real lam0
      common/blck2/at(300,300),iat,jat
      common/blck3/ata(300,300),iata,jata
      common/blck13/d(300),id
      common/blck15/dp(300),pnew(300),mnew(300),dnew(300)
      common/blck29/atd(300)
      common/blck31/aug(300,600),iaug,jaug
c---set up term [(at)(a) + (lam)(id)]
      do 10 i=1,iata
      do 10 j=1,jata
      aug(i,j)=ata(i,j)
      if (i.eq.j) aug(i,j)=ata(i,j)+lam0
 10   continue
      iaug=iata
      jaug=jata
c---create augmented matrix in preparation for gelim
      do 3 i=1,iat
    3 atd(i)=0.
      do 4 i=1,iat
      do 4 j=1,id
    4 atd(i)=atd(i)+at(i,j)*d(j)
      iatd=iat
      jatd=1
```

```
      call augm(iatd,jatd)
c---solve for dp
      call gelim
      do 20,i=1,iatd
      dp(i)=aug(i,jaug)
 20   continue
      return
      end
c
c-------------------------------------------------------------------------c
c   This subroutine to solve a matrix equation of the form                c
c                                                                         c
c               ac=b                           (1)                       c
c                                                                         c
c   where c is unknown.                                                   c
c   it is assumed that a is a square matrix with dimensions ia x ia.      c
c   b is a matrix (or vector) with dimensions ia x jb.                    c
c   input matrix aug is an augmented matrix consisting of matrices        c
c   a and b, i.e.                                                         c
c                                                                         c
c                                                                         c
c               [aug]=[a:b]                    (2)                       c
c                                                                         c
c   where aug has dimensions ia x ia+jb.  the solution, c, to equation    c
c   (1) is returned in the last jb columns of aug.                        c
c                                                                         c
c   note: to obtain the inverse of a matrix a simply make b in (2)        c
c   an identity matrix with the same dimensions as a.  gelim will         c
c   then solve the equation                                               c
c                                                                         c
c               ac=i                            (3)                      c
c                                                                         c
c   the solution of (3) is a-inverse and will be returned                 c
c   in the last ia columns of aug.                                        c
c-------------------------------------------------------------------------c
      subroutine gelim
      common/blck31/aug(300,600),iaug,jaug
      ia=iaug
      jb=iaug
      do 80 i=1,ia
      piv=0.0
      do 30 j=i,ia
      if (abs(aug(j,i)).le.piv) go to 30
      piv=abs(aug(j,i))
      ip=j
 30   continue
      if (piv.gt.0.0) go to 40
      write(*,31)i
 31   format(1x,'Error. Singular matrix! Check row',i3)
      stop
 40   if (ip.eq.i) go to 55
      do 50 j=1,ia+jb
      hold=aug(i,j)
```

```
      aug(i,j)=aug(ip,j)
      aug(ip,j)=hold
 50   continue
 55   c=aug(i,i)
      do 60 j=i,ia+jb
      aug(i,j)=aug(i,j)/c
 60   continue
      do 80 j=1,ia
      if (j.eq.i) go to 80
      c=aug(j,i)
      do 70 k=i,ia+jb
      aug(j,k)=aug(j,k)-c*aug(i,k)
 70   continue
 80   continue
      return
      end
c
c----------------------------------------------------------------------c
c   This routine creates an augmented matrix as follows:               c
c                                                                      c
c                           [aug]=[a:b]                                c
c                                                                      c
c   note:  the matrix a is placed in aug prior to the subroutine call  c
c          and passed with its dimensions through common blck31.  this c
c          subroutine merely places the ib x jb matrix b in the        c
c          next jb columns of aug.                                     c
c----------------------------------------------------------------------c
      subroutine augm(ib,jb)
      common/blck29/atd(300)
      common/blck31/aug(300,600),iaug,jaug
      j1=jaug+1
      do 100,i=1,iaug
      do 100,j=jaug+1,jaug+jb
      if(j.eq.j1) aug(i,j)=atd(i)
      if(j.ne.j1) aug(i,j)=0.0
 100  continue
      jaug=jaug+jb
      return
      end
c
c----------------------------------------------------------------------c
c   Subroutine printm to print the values of an ia x ja matrix a.      c
c   Arguments:
c             mi   option of the data to be printed                    c
c             iop  unit number of open                                 c
c----------------------------------------------------------------------c
      subroutine printm(mi,iop)
      dimension tp(300)
      common/blck11/m(300),im
      common/blck12/o(300),io
      common/blck14/p(300),ip
      common/geov2/td,xlen,ext,z0,nf,xb(25),lay(25)
      common/mden/den
```

```
      nf2=nf*2
      if(mi.eq.11) then
      write(iop,1000) (m(i),i=1,im)
      return
      endif
      if(mi.eq.12) then
      write(iop,1000) (o(i),i=1,io)
      return
      endif
      if(mi.eq.14) then
      do 1 i=1,ip
      tp(i)=p(i)
      if((i.gt.nf2).and.(mod(i-nf2,3).eq.0)) tp(i)=tp(i)+den
   1  continue
      write(iop,1000) (tp(i),i=1,ip)
      return
      endif
      print *,'print wrong'
 1000 format(6f10.2)
      return
      end
c
c----------------------------------------------------------------------c
c   Subroutine addm to add p and dp to generate pnew                   c
c----------------------------------------------------------------------c
      subroutine addm
      integer fixp
      common/blck32/fixp(300),if
      common/blck14/p(300),ip
      common/blck15/dp(300),pnew(300),mnew(300),dnew(300)
      common/pscale/delp(5),afa1(5),npf,afa(300)
      common/layer1/ilay,lblk(25),npar(8,25)
      common/geov2/td,xlen,ext,z0,nf,xb(25),lay(25)
      common/box/xbl,zbl,dbl,xbt,zbt,dbt,xbr,zbr,dbr,xbd,zbd,dbd
      common/fault/txf(25)
c-----adding corrections----------------------------------------------c
      do 1 i=1,ip
c    1 pnew(i)=p(i)+dp(i)
   1  pnew(i)=p(i)+dp(i)*afa(i)
      ix=1
c-----constrain faults------------------------------------------------c
      xlft=xbl
      npf1=npf-1
      do 6 i=1,npf1,2
      call xpoint(pnew(i),z0,pnew(i+1),xbd,zbd,dbd,xp,zp)
      if(i.eq.npf1) goto 13
      call xpoint(pnew(i+2),z0,pnew(i+3),xbd,zbd,dbd,xrht,zrht)
  13  if((xb(ix).gt.pnew(i)).or.(xb(ix+1).lt.pnew(i))) then
       pnew(i)=p(i)
       afa(i)=afa(i)/5.
      endif
      if((xlft.gt.xp).or.(xrht.lt.xp)) then
       pnew(i+1)=p(i+1)
```

```
       afa(i+1)=afa(i+1)/5.
      endif
      if(i.eq.npf1) goto 14
      if(pnew(i).gt.pnew(i+2)) then
      pnew(i)=p(i)
      pnew(i+2)=p(i+2)
      afa(i)=afa(i)/5.
      afa(i+2)=afa(i+2)/5.
      endif
  14  ix=ix+1
      xlft=xp
   6  continue
c-----constrain model surface be unchanged-----------------------------c
      n2f=2*nf
      nlay=0
      do 2 i=1,nf
      ilfz=n2f+1+nlay
      ilfd=ilfz+1
      pnw=pnew(2*i-1)
      irtz=ilfz+3*lay(i)
      irtd=ilfd+3*lay(i)
      if(((fixp(ilfz).eq.1).and.(fixp(ilfd).eq.1)).and.
     *((fixp(irtz).eq.1).and.(fixp(irtd).eq.1))) then
      call xpoint(xb(i),p(ilfz),p(ilfd),p(2*i-1),z0,p(2*i),xp,zp)
      dxp=xp-pnw
      dzp=zp-z0
      if((dxp.eq.0.).or.(dzp.eq.0.)) goto 21
      fdip=atan(abs(dzp/dxp))
      if(xp.ge.pnw) pnew(2*i)=fdip*57.29578
      if(xp.lt.pnw) pnew(2*i)=180.-fdip*57.29578
      adf=abs(pnew(2*i)-p(2*i))
      if(adf.le.5.) goto 29
      pnew(2*i-1)=p(2*i-1)
      afa(2*i-1)=afa(2*i-1)/5.
      pnew(2*i)=p(2*i)
      afa(2*i)=afa(2*i)/5.
  29  endif
      goto 22
  21  pnew(2*i-1)=p(2*i-1)
      pnew(2*i)=p(2*i)
  22  nlay=3*lay(i)+nlay
   2  continue
      id=npf
c-----constrain depths-------------------------------------------------c
      do 7 i=1,ix
      dt=z0
      do 8 j=1,lay(i)-1
      idp=id+1
      if(pnew(idp).lt.dt) then
        pnew(idp)=p(idp)
        afa(idp)=afa(idp)/5.
      endif
      if(pnew(idp).gt.pnew(idp+3)) then
```

```
        pnew(idp)=p(idp)
        afa(idp)=afa(idp)/5.
      endif
      id=idp+2
      dt=pnew(idp)
    8 continue
      id=id+3
    7 continue
      ic=npf
c-----constrain continuity of bedding and constant throw---------------c
      do 9 i=1,nf
      if(txf(i).ge.100.) goto 11
      k=i*2
      do 10 j=1,lay(i)-1
      il=ic+2
      ir=il+3*lay(i)
      call xpoint(xb(i),pnew(il-1),pnew(il),pnew(k-1),z0,pnew(k),xp,zp)
      if(txf(i).eq.0.) goto 12
      zp=zp+txf(i)
      if(pnew(k).eq.0.) goto 12
      pwn=pnew(k)*0.0174532
      xp=xp+(txf(i)/tan(pwn))
c   12  cr=atan((zp-pnew(ir-1))/(xp-xb(i+1)))
c       pnew(ir)=cr*57.29578
   12 call xpoint(xp,zp,pnew(ir),xb(i+1),z0,90.0,xp1,zp1)
      pnew(ir-1)=zp1
      ic=il+1
   10 continue
      ic=ic+3
      goto 9
   11 ic=ic+3*lay(i)
    9 continue
c-----constrain continuity of layer densities--------------------------c
      if(ilay.ge.1) then
      do 3 i=1,ilay
      adp=0.
c-----calculate the average density corrections------------------------c
      do 4 j=1,lblk(i)
      j1=npar(i,j)
    4 adp=adp+dp(j1)/lblk(i)
      nfix=npar(i,1)
      pnew(nfix)=p(nfix)+adp
      do 5 j=2,lblk(i)
      k=npar(i,j)
    5 pnew(k)=pnew(nfix)
    3 continue
      endif
      return
      end
```

```
c----------------------------------------------------------------------c
c     This routine is to set layering parameters called from clips     c
c     as submit running                                                c
c     Arguments:  cn      layering order number                        c
c                 cbk     order number in layering cn                  c
c                 cb      block number in the model                    c
c                 cl      layer number in block cb                     c
c     Variables:  nfix    order number of parameter                    c
c----------------------------------------------------------------------c
      subroutine sublay (cn,cbk,cb,cl)
      common/layer1/ilay,lblk(25),npar(8,25)
      common/layer2/nfix
      real cn,cbk,cb,cl
      icn=cn
      inb=cbk
      icb=cb
      icl=cl
      ilay=icn
      if(inb.gt.lblk(icn)) lblk(icn)=inb
      call fixpf(icb,icl)
      npar(icn,inb)=nfix
      return
      end
```

```
c----------------------------------------------------------------------c
c     This subroutine is to read and set facts from file cname         c
c     called from clips                                                c
c     Arguments:  cname    filename storing the facts                  c
c----------------------------------------------------------------------c
      subroutine fkread(cname)
      integer cname,cstr(18)
      character fname*10,fstr*72
      call loadc(fname,cname)
      open(11,file=fname)
    1 read(11,'(a72)',end=99) fstr
      call storec(fstr,cstr)
      call assert(cstr)
      goto 1
   99 return
      end
```

```
c----------------------------------------------------------------------c
c     This subroutine is to set fixed parameters called from clips     c
c     Arguments:  cfob     fault or bed (f or b)                       c
c                 ctp      type of parameter (i.e. x, z, or d)         c
c                 cbof     number of fault or block                    c
c                 cbd      number of bed                               c
c----------------------------------------------------------------------c
      subroutine fixpar(cfob,ctp,cbof,cbd)
      integer cfob,ctp
      real cbof,cbd
```

```
      character fob,type
      common/layer/nf,nsk(25)
      call loadc(fob,cfob)
      call loadc(type,ctp)
      nbof=cbof
      nbd=cbd
      ns=2*nf
      if(fob.eq.'f') then
        if(type.eq.'x') nfix=2*nbof-1
        if(type.eq.'d') nfix=2*nbof
        goto 99
      else
        do 1 i=2,nbof
 1      ns=3*nsk(i-1)+ns
        do 2 j=1,nbd-1
 2      ns=ns+3
        if(type.eq.'z') nfix=ns+1
        if(type.eq.'d') nfix=ns+2
        if(type.eq.'r') nfix=ns+3
      endif
 99   return(nfix)
      end

c-------------------------------------------------------------------c
c     This subroutine is to set fault throw called from clips       c
c     Arguments:  fn    fault number                                c
c                 ft    throw value                                 c
c-------------------------------------------------------------------c
      subroutine fthrow(fn,ft)
      real fn,ft
      common/fault/txf(25)
      nf=fn
      txf(nf)=ft
      return
      end


c-------------------------------------------------------------------c
c     This subroutine is to modify weight called from clips         c
c     Arguments:   iw     parameter order number                    c
c                   w     new weighting factor value                c
c-------------------------------------------------------------------c
      subroutine modwt(iw,w)
      real iw,w
      common/pscale/delp(5),afai(5),npf,afa(300)
      niw=iw
      afa(niw)=w
      return
      end


c-------------------------------------------------------------------c
c     This subroutine is to set facts of the layering for being used c
```

```
c     to build advanced rules                                          c
c     Arguments: lay    layering order                                 c
c                bloc   number of blocks in the layering lay           c
c----------------------------------------------------------------------c
      subroutine slay(lay,bloc)
      common/layer1/ilay,lblk(25),npar(8,25)
      common/layer2/nfix
      real lay,bloc
      character*4 cb,cnb,cl,ci
      character*32 blay
      integer ibl(8)
      nlay=lay
      nblk=bloc
      ilay=nlay
      lblk(nlay)=nblk
   99 write(*,*) "enter each order numbers of block and layer  "
      write(*,*) "of the ",nblk, " bodies in the layer # ",nlay," :"
      write(ci,'(i4)') nlay
      do 1 i=1,nblk
      write(*,*) "body #",i
      read(*,*,err=99) nb,nl
      call fixpf(nb,nl)
      npar(nlay,i)=nfix
      write(cb,'(i4)') nb
      write(cl,'(i4)') nl
      write(cnb,'(i4)') i
      blay='lay'//ci//cnb//cb//cl
      call storec(blay,ibl)
      call assert(ibl)
    1 continue
      return
      end
```

```
c----------------------------------------------------------------------c
c     This subroutine is to set facts of initial model for being used  c
c     to build more advanced rules to control the layering problems    c
c     called from clips                                                c
c     Arguments:  cmod    initial model filename                       c
c----------------------------------------------------------------------c

      subroutine slook(cmod)
      integer cmod
      character*11 fmod
      character*5 ci,clay,cj
      character*10 ctd,clen,cext,cz0,cnf,cxb
      character*10 cxf,cdf,cdb,czb,crb,cden
      character*28 ct,cl,ce,cz,cn,cdm
      character*32 cx,cfx,cfd,cbz,cbd,cbr
      integer ict(7),icl(7),ice(7),icz(7),icn(7),icx(8)
      integer icfx(8),icfd(8),icbz(8),icbd(8),icbr(8),icd(7)
      common/layer/nf,nsk(25)
```

```
      common/fault/txf(25)
      common/mden/den
      call loadc(fmod,cmod)
      open(11,file=fmod,blank='null')
      read(11,*) td,xlen,ext,z0,nf,den
      write(ctd,'(f10.2)') td
      write(clen,'(f10.2)') xlen
      write(cext,'(f10.2)') ext
      write(cz0,'(f10.2)') z0
      write(cnf,'(i10)') nf
      write(cden,'(f10.2)') den
      do 4 i=1,nf
    4 txf(i)=100.
      ct='depth '//ctd
      cl='width '//clen
      ce='extent '//cext
      cz='top '//cz0
      cn='num-fault '//cnf
      cdm='mean-density '//cden
      call storec(ct,ict)
      call storec(cl,icl)
      call storec(ce,ice)
      call storec(cz,icz)
      call storec(cn,icn)
      call storec(cdm,icd)
      call assert(ict)
      call assert(icl)
      call assert(ice)
      call assert(icz)
      call assert(icn)
      call assert(icd)
      do 1 i=1,nf
      read(11,*) xfib,dfib
      write(cxf,'(f10.2)') xfib
      write(cdf,'(f10.2)') dfib
      write(ci,'(i5)') i
      cfx='fault x '//ci//cxf
      cfd='fault d '//ci//cdf
      call storec(cfx,icfx)
      call storec(cfd,icfd)
      call assert(icfx)
      call assert(icfd)
    1 continue
      nf1=nf+1
      do 2 i=1,nf1
      read(11,*) xbo,lyo
      nsk(i)=lyo
      write(cxb,'(f10.2)') xbo
      write(clay,'(i5)') lyo
      write(ci,'(i5)') i
      cx='section '//ci//cxb//clay
      call storec(cx,icx)
      call assert(icx)
```

```
      do 3 j=1,lyo
      read(11,*) zbib,dbib,rbib
      write(czb,'(f10.2)') zbib
      write(cdb,'(f10.2)') dbib
      write(crb,'(f10.4)') rbib
      write(cj,'(i5)') j
      cbd='bed d '//ci//cj//cdb
      cbz='bed z '//ci//cj//czb
      cbr='bed r '//ci//cj//crb
      call storec(cbd,icbd)
      call storec(cbz,icbz)
      call storec(cbr,icbr)
      call assert(icbd)
      call assert(icbz)
      call assert(icbr)
    3 continue
    2 continue
      close(11)
      return
      end
```

```
c----------------------------------------------------------------c
c    This subroutine is to calculate the order of parameter called  c
c    from fortran                                                   c
c    Arguments:  nb    block order number                           c
c                nl    layer order in block nb                      c
c----------------------------------------------------------------c
      subroutine fixpf(nb,nl)
      common/layer/nf,nsk(25)
      common/layer2/nfix
      ns=2*nf
      do 1 i=2,nb
    1 ns=ns+3*nsk(i-1)
      do 2 j=1,nl
    2 ns=ns+3
      nfix=ns
      return
      end

c--------------------------------------------------------------------c
c   This subroutine is to find the maximum ( m > 0 ) or minimum ( m < 0 ) c
c   value of an array                                                   c
c   Arguments:  lx    dimension of x array                              c
c                x    array of data                                     c
c             xmax    the extreme value returned                        c
c                m    option factor                                     c
c--------------------------------------------------------------------c

      subroutine maxmn(lx,x,xmax,m)
      dimension x(lx)
      id=1
      do 1 i=1,lx
      if(m.ge.0) goto 2
      if(x(id).gt.x(i)) id=i
      goto 1
    2 if(x(id).lt.x(i)) id=i
    1 continue
      xmax=x(id)
      return
      end

c--------------------------------------------------------------------c
c   This subroutine is to create a observed data file called from clips  c
c   Arguments:  file    observations filename                           c
c--------------------------------------------------------------------c

      subroutine creato(file)
      integer file
      character nfile*8,ans*1
      common/gpars/fx(300),fz(300),grav(300),ograv(300),npts,gfac
      call loadc(nfile,file)
      open(11,file=nfile)
      rewind(11)
      print *, 'How many observed data ?'
```

```
      read(*,*) npts
      print *, 'Are there',npts,' observed data ? (y/n)'
      read(*,'(a)') ans
      if((ans.eq.'y').or.(ans.eq.'Y')) goto 31
      print *,'What is the correct number ?'
      read(*,*) npts
   31 write(11,101) npts
      print *, ' Are all data on the same level ? (y/n)'
      read(*,'(a)') ans
      print *,' *** Note: z-coordinates are positive downward '
      print *,'           x-coordinates are positive rightward'
      print *,'           unit is kilometer'
      if((ans.eq.'n').or.(ans.eq.'N')) goto 20
      print *, ' What is the z value of the datum ? '
      read(*,*) fz0
      do 1 i=1,npts
    1 fz(i)=fz0
      goto 21
   20 print *,' Please enter the z-coordinate for each point'
      do 2 i=1,npts
      print *, 'z... #',i,' = '
    2 read(*,*) fz(i)
   21 print *,' Are the points equally spaced ? (y/n)'
      read(*,'(a)') ans
      if((ans.eq.'n').or.(ans.eq.'N')) goto 22
      print *,' What is the space interval ?'
      read(*,*) fi
      print *,' What is the x-coordinate to start from ?'
      read(*,*) fs
      do 3 i=1,npts
    3 fx(i)=fs+(i-1)*fi
      goto 23
   22 print *,' Please enter the x-coordinate for each point'
      do 4 i=1,npts
      print *,'x... #',i,' = '
    4 read(*,*) fx(i)
   23 print *,' Please enter the observed data for each point'
      do 5 i=1,npts
      print *,' obs... #',i,' = '
    5 read(*,*) ograv(i)
   25 write(*,102)
      k=0
      do 6 i=1,npts
      k=k+1
      write(*,103) i,fx(i),fz(i),ograv(i)
      if(k.le.20) goto 6
      print *,' Hit return key to continue'
      read(*,'(a)') ans
      k=0
    6 continue
      print *,'Are all data correct ? (y/n)'
      read(*,'(a)') ans
      if((ans.eq.'n').or.(ans.eq.'N')) goto 24
```

```
      write(11,*) (fx(i),fz(i),ograv(i),i=1,npts)
      goto 99
   24 print *,'How many points are to be corrected ?'
      read(*,*) ip
      do 7 i=1,ip
      print *,'Which point needs to be corrected ?'
      read(*,*) ic
      write(*,102)
      write(*,103) ic,fx(ic),fz(ic),ograv(ic)
      print *,'Enter the correct x,z,and obs values seperated by space'
    7 read(*,*) fx(ic),fz(ic),ograv(ic)
      goto 25
   99 close(11)
  101 format(i5)
  102 format(1x,'no.',5x,'-x-',5x,'-z-',5x,'-g-',/,1x,'----------------
     *--------------------------')
  103 format(1x,i3,3x,f7.2,3x,f7.2,3x,f7.2)
      return
      end

c----------------------------------------------------------------------c
c    This subroutine is to create an initial model called from clips      c
c    Arguments:  file    initial model filename                          c
c----------------------------------------------------------------------c
      subroutine creatm(mfile)
      integer mfile
      character nfile*8
      common/ibod/xfib(25),dfib(25),zbib(25,8),dbib(25,8),rbib(25,8)
      common/mden/den
      common/geov2/td,xlen,ext,z0,nf,xb(25),lay(25)
      call loadc(nfile,mfile)
      open(12,file=nfile)
      rewind(12)
      print *,'What is the total depth of the model ?'
      read(*,*) td
      print *,'What is the total width of the model ?'
      read(*,*) xlen
      print *, 'How much extension would be added to the model ?'
      read(*,*) ext
      print *, 'What is the top level of the model ?'
      read(*,*) z0
      print *, 'How many faults in this model ?'
      read(*,*) nf
      print *, 'What is the mean density ?'
      read(*,*) den
      write(12,*) td,xlen,ext,z0,nf,den
      do 1 i=1,nf
      print *, 'What is the x-coordinate and dip angle of fault #',i
      read(*,*) xfib(i),dfib(i)
    1 write(12,*) xfib(i),dfib(i)
      do 2 i=1,nf+1
      print *,'What is the x-fixed coordinate of block #',i
      read(*,*) xb(i)
```

```
      print *, 'How many layers in block #',i
      read(*,*) lay(i)
      write(12,*) xb(i),lay(i)
      do 3 j=1,lay(i)
      print *,'Enter z, dip, and density values of bed #',j
      read(*,*) zbib(i,j),dbib(i,j),rbib(i,j)
    3 write(12,*) zbib(i,j),dbib(i,j),rbib(i,j)
    2 continue
      close(12)
      return
      end

c-----------------------------------------------------------------------------c
c     This routine loads a fortran character string from a c language         c
c     character string.  "cstring" is the c character in a fortran integer    c
c     variable; "chars" is the fortran character variable into which the      c
c     the string is to be stored.                                             c
c-----------------------------------------------------------------------------c
      subroutine loadc(chars,cstring)
      integer cstring(1)
      character*1 char
      character*(*) chars
      iword=1
      ivalue=0
      chars=''
      ibit=24
      k=len(chars)
      do 100 i=1,k
      call mvbits(cstring(iword),ibit,8,ivalue,0)
      if(ivalue .lt.32) goto 200
      chars(i:i)=char(ivalue)
      ibit=ibit-8
      if(ibit .ge. 0) goto 100
      ibit=24
      iword=iword+1
  100 continue
  200 continue
      return
      end

c------------------------------------------------------------------------c
c     This routine converts a fortran character string into a c language c
c     character string.  "chars" is the input character string;          c
c     "cstring" is the c character string in a fortran integer variable. c
c------------------------------------------------------------------------c
      subroutine storec(chars,cstring)
      integer cstring(1)
      character*(*) chars
      iword=1
      ibit=24
      k=length(chars)
      do 100 i=1,k
      ivalue=ichar(chars(i:i))
```

```
      call mvbits(ivalue,0,8,cstring(iword),ibit)
      ibit=ibit-8
      if(ibit .ge. 0) goto 100
      ibit=24
      iword=iword+1
  100 continue
      call mvbits(0,0,8,cstring(iword),ibit)
      return
      end
      integer function length(string)
      character*(*) string
      k=len(string)
      do 100 i=k,1,-1
      if(string(i:i).ne.'') goto 150
  100 continue
  150 continue
      length=i
      return
      end

c-------------------------------------------------------------------------c
c     This routine works as vax mvbits function for shifting bits         c
c-------------------------------------------------------------------------c
      subroutine mvbits(m,i,l,n,j)
c
c     routine to emulate vax mvbits
c
c     m:   integer
c          source array for bit field
c
c     i:   integer
c          first bit position to be transferred from m
c
c     l  : integer
c          number of bits to be transferred from m
c
c     n:   integer
c          target array for bit field
c
c     j:   integer
c          first bit position to be filled in n
c
c     this emulation assumes that integers are stored as 32-bit
c     bit arrays with least significant bit as unit.
c
      integer m,i,l,n,j
c
      integer mbit,nbit,ibit,ivalue,jvalue
c
c     check for nothing to do
c
      if (l .le. 0) return
      if ((i .lt. 0) .or. (i .ge. 32)) return
```

```
      if ((j .lt. 0) .or. (j .ge. 32)) return
c
      mbit = i
      nbit = j
      ibit = 1
c
c     shift bit to last bit
c
  100 ivalue = rshift(m,mbit)
c
c     now shift bit to destination location
c
      ivalue = lshift(ivalue,nbit)
c
c     mask source array and zap bit in target array
c
      jvalue = 1
      jvalue = lshift(jvalue,nbit)
      ivalue = and(ivalue,jvalue)
      jvalue = not(jvalue)
      n = and(n,jvalue)
c
c     insert bit in target array
c
      n = or(n,ivalue)
c
c     increment bits
c
      ibit = ibit + 1
      if (ibit .gt. l) return
      mbit = mbit + 1
      if (mbit .ge. 32) return
      nbit = nbit + 1
      if (nbit .ge. 32) return
c
c     loop back for another bit
c
      go to 100
c
      end
```

```
c-----------------------------------------------------------------------c
c     This subroutine is to plot the observed and calculate data and   c
c     the model                                                         c
c     Arguments:  obsfil     observed data filename                     c
c                 modfil     model filename                             c
c                 calfil     model response filename                    c
c                 rmsi       root mean square error between             c
c                            observations and calculations              c
c-----------------------------------------------------------------------c

      subroutine graph(rmsi)
      dimension sx(300),sy(300)
      character crms*7,srms*12
c     character*1 p
      character*32 outfil
      character*5 lbl1,lbl2,lbl3,cden
      common/box/xbl,zbl,dbl,xbt,zbt,dbt,xbr,zbr,dbr,xbd,zbd,dbd
      common/geov1/xf(25),dipf(25),zb(25,8),dipb(25,8),rhob(25,8)
      common/gpars/fx(300),fz(300),grav(300),ograv(300),npts,gfac
      common/geov2/td,xlen,ext,z0,nf,xb(25),lay(25)
      common/rmse/crms
      common/mden/den
c-----graphics size variables
      outfil='outgf'
      xln=7.25
      yht=5.625
      xleft=0.875
      ylow=0.5
      xrite=xleft+xln
      ytop=ylow+yht
      datah=2.5
      dph=2.0
      skip=yht-datah-dph
      tic=0.1
      nf1=nf+1
      call maxmn(npts,ograv,ymax1,1)
      call maxmn(npts,grav,ymax2,1)
      if(ymax1.ge.ymax2) goto 30
      ymax=ymax2
      goto 31
   30 ymax=ymax1
   31 call maxmn(npts,ograv,ymin1,-1)
      call maxmn(npts,grav,ymin2,-1)
      if(ymin1.le.ymin2) goto 32
      ymin=ymin2
      goto 33
   32 ymin=ymin1
   33 xmin=-ext
      xmax=xlen+ext
      dy=ymax-ymin
      dx=xmax-xmin
      ymaxg=ymax+dy*.1
      yming=ymin-dy*.1
```

```
      dyg=ymaxg-yming
      rx=xln/dx
      ry=datah/dyg
c-----disappear the cursor prompt
      call escape()
      call literal( "G" )
      call literal( "a" )
      call literal( "c" )
      call vpfile(outfil)
      call vperase
      call vpcolor(6)
      call vpfat(2)
c-----draw axis and box
      call vpmove(xleft,ytop)
      y1=ytop-datah
      call vpdraw(xleft,y1)
      call vpdraw(xrite,y1)
      y2=y1-skip
      call vpmove(xrite,y2)
      call vpdraw(xleft,y2)
      y3=y2-dph
      call vpdraw(xleft,y3)
      call vpdraw(xrite,y3)
      call vpdraw(xrite,y2)
      call xytic(dx,dlpx)
c-----draw tics and scale for x axis
      np=0
      x1=xmin-mod(xmin,dlpx)
      x2=dlpx*int((xmax-mod(xmax,dlpx))/dlpx)
      do 100 t1=x1,x2,dlpx
      np=np+1
      ntic=mod(np,2)
      nt1=int(t1)
      write(lbl1,'(i5)') nt1
      t2=(t1+ext)*rx+xleft
      t3=t2-0.125
      y1tk=y1-tic
      y2tk=y2+tic
      y1lb=y1-0.25
      call vpmove(t2,y1)
      call vpdraw(t2,y1tk)
      call vpmove(t2,y2tk)
      call vpdraw(t2,y2)
      if(ntic.ne.1) goto 100
      call vptext(t3,y1lb,3,0,lbl1)
  100 continue
c-----draw tics and scale for y axis
      call xytic(dyg,dlpy)
      y1g=int(yming/dlpy)*dlpy
      y2g=int(ymaxg/dlpy)*dlpy
      if(y1g.gt.0.) y1g=y1g+dlpy
      if(y2g.gt.0.) y2g=y2g+dlpy
      np=0
```

```
      do 200 t=y1g,y2g,dlpy
      tx=xleft-0.375
      np=np+1
      ntic1=mod(np,2)
      nt=int(t)
      if(t.lt.ymaing.or.t.gt.ymaxg) goto 200
      ty=(t-ymaing)*ry+y1
      xtk=xleft-tic
c      if(nt.gt.0) nt=nt+1
      write(lbl2,'(i5)') nt
      call vpmove(xleft,ty)
      call vpdraw(xtk,ty)
      if(ntic1.ne.1) goto 200
      ty1=ty-0.01
      call vptext(tx,ty1,3,0,lbl2)
  200 continue
      xm=xleft-0.05
      ym=ytop+0.15
      call vpcolor(7)
      call vptext(xm,ym,3,0,'mGal')
      call vpcolor(2)
      do 41 i=1,npts
      sx(i)=(fx(i)+ext)*rx+xleft
   41 sy(i)=(ograv(i)-ymaing)*ry+y1
      call vpmove(sx(1),sy(1))
c      call vpsetdash(0.07,0.05,1)
      call vppline(sx,sy,npts)
      do 42 i=1,npts
   42 sy(i)=(grav(i)-ymaing)*ry+y1
      call vpcolor(6)
c      call vpsetdash(0.01,0.0,1)
      call vpmove(sx(1),sy(1))
      call vppline(sx,sy,npts)
c-----td1 is model scaled depth
c-----rt is enlarged rate
c-----td is real model depth
c-----top is original top
c-----bot is original bottom
c-----z0 is real model top
c-----top1 is scaled top
c-----bot1 is scaled bottom
c-----bot is enlarged bottom on graph
      td1=dph
      rt=td1/td
      top1=y2
      bot1=y3
      if(td.gt.100.) dlpz=50.
      if(td.le.100.) dlpz=20.
      if(td.le.50.) dlpz=10.
      if(td.le.25.) dlpz=5.
      if(td.le.10.) dlpz=2.
      if(td.le.5.) dlpz=1.
      dpz=-dlpz*rt
```

```
      np=0
      do 7 t=top1,bot1,dpz
      tt=dlpz*np
      np=np+1
      ntt=int(tt)
      zx=xleft-0.375
      write(lbl3,'(i5)') ntt
      xtdp=t
      call vpmove(xleft,xtdp)
      call vpdraw(xtk,xtdp)
      tz1=t-0.01
      call vptext(zx,tz1,3,0,lbl3)
    7 continue
      call vpcolor(7)
      xtt=xleft-0.04
      tt33=y3-0.2
      call vptext(xtt,tt33,3,0,'Km')
      xl1=xleft+1.5
      xl2=xl1+2.2
      yy1=y1-0.625
      yy2=ytop+0.0625
      call vptext(xl2,yy1,3,0,'Distance (Km)')
      call vpcolor(2)
      call vptext(xl1,yy2,3,0,'----: Observed')
      call vpcolor(6)
      call vptext(xl2,yy2,3,0,'----: Calculated')
      write(crms,'(f7.3)') rms1
      srms='RMSE:'//crms
      xl3=xl2+2.2
      call vpcolor(7)
      call vptext(xl3,yy2,3,0,srms)
      call vpcolor(2)
      top=z0
      bot=top-td
      xlf=xbl
      zlf=top
      dlf=dbl
      zrt=top-zbr
      drt=dbr
      do 2 i=1,nf
      dipi=180.-dipf(i)
      call xpoint(xf(i),top,dipi,0.,bot,0.,xp,zp)
      if(xp.ge.xlf) goto 11
      call xpoint(xf(i),top,dipi,xlf,zlf,dlf,xp,zp)
      goto 12
   11 if(xp.le.xbr) goto 12
      call xpoint(xf(i),top,dipi,xbr,zrt,drt,xp,zp)
   12 xlf=xp
      dlf=180.-dipf(i)
      xp=xleft+(ext+xp)*rx
      xfi=xleft+(ext+xf(i))*rx
      zp=top1+zp*rt
      call vpmove(xfi,top1)
```

```
2  call vpdraw(xp,zp)
   zd=top-zbd
   dd=dbd
   do 3 i=1,nf1
   xtop1=xf(i-1)
   ztop1=top
   xtp1=xf(i-1)
   ztp1=top
   xtp2=xf(i)
   ztp2=top
   xtp=xbt
   ztp=top
   dtp=dbt
   do 3 j=1,lay(i)
   if(i.eq.1) xtp1=xbl
   if(i.eq.1) xtop1=xbl
   if(i.eq.nf1) xtp2=xbr
   if(i.eq.nf1) xtop2=xbr
   xl=xf(i-1)
   zl=top
   dl=180.-dipf(i-1)
   xr=xf(i)
   zr=top
   dr=180.-dipf(i)
   if(i.ne.1) goto 21
   xl=xbl
   dl=dbl
   goto 22
21 if(i.ne.nf1) goto 22
   xr=xbr
   dr=dbr
22 dpij=-dipb(i,j)
   zbij=top-zb(i,j)
   call xpoint(xl,zl,dl,xb(i),zbij,dpij,xp1,zp1)
   call xpoint(xr,zr,dr,xb(i),zbij,dpij,xp2,zp2)
   if(zp1.le.ztp1) goto 13
   call xpoint(xtp,ztp,dtp,xb(i),zbij,dpij,xp1,zp1)
   goto 14
13 if(zp2.le.ztp2) goto 14
   call xpoint(xtp,ztp,dtp,xb(i),zbij,dpij,xp2,zp2)
14 if(j.eq.lay(i)) goto 16
   if(zp1.gt.zd) goto 15
   call xpoint(xbd,zd,dd,xb(i),zbij,dpij,xp1,zp1)
   goto 16
15 if(zp2.gt.zd) goto 16
   call xpoint(xbd,zd,dd,xb(i),zbij,dpij,xp2,zp2)
16 xtp=xb(i)
   ztp=top-zb(i,j)
   dtp=-dipb(i,j)
   xpr=xleft+(ext+(1.25*xtp1+xtp2+1.25*xp1+xp2)/4.5)*rx
   zpr=(2*ztp1+2*ztp2+3*zp1+3*zp2)/10.
   zpr=top1+zpr*rt
   write(cden,'(f5.3)') rhob(i,j)+den
```

```
      call vptext(xpr,zpr,3,0,cden)
      ztp1=zp1
      ztp2=zp2
      xtop1=xp1
      ztop1=zp1
      zp1=top1+zp1*rt
      zp2=top1+zp2*rt
      xp1=xleft+(ext+xp1)*rx
      xp2=xleft+(ext+xp2)*rx
      call vpmove(xp1,zp1)
    3 call vpdraw(xp2,zp2)
      call vpendplot
      call psystem
c      read(*,'(a)') p
      call escape()
      call literal( "G" )
      call literal( "c" )
      return
      end
c-----------------------------------------------------------------c
c     This routine is to set a reasonable scale interval for axis c
c-----------------------------------------------------------------c
      subroutine xytic(val,xyint)
      dimension x(4)
      x(1)=16.
      x(2)=40.
      x(3)=80.
      if(val.gt.1.) goto 9
      xyint=0.5
      goto 99
  9   do 1 i=1,3
      if(val.gt.x(i)) goto 1
      xyint=x(i)/16.
      goto 99
  1   continue
      do 2 j=1,3
  2   x(j)=x(j)*10.
      goto 9
  99  return
      end
```

APPENDIX B

Listing of C Interfaces Between CLIPS and FORTRAN

```
/*  CLIPS Version 4.01  06/12/87 */

#include <stdio.h>
#include "clips.h"

/*************************************************************/
/* MAIN: Start execution of CLIPS.  This function must be      */
/*   redefined in order to embed CLIPS within another program. */
/*   Example of redefined main:                                */
/*     main()                                                  */
/*       {                                                     */
/*        init_clips();                                        */
/*            .                                                */
/*            .                                                */
/*            .                                                */
/*        process_data();                                      */
/*        run(-1);                                             */
/*        evaluate_data();                                     */
/*            .                                                */
/*            .                                                */
/*            .                                                */
/*        final_results();                                     */
/*       }                                                     */
/*************************************************************/
MAIN__()
  {
   init_clips();
   cl_print("wclips","        CLIPS (V4.01 06/12/87)\n");
   command_loop();
  }

/*************************************************************/
/* USRFUNCS:  The function which informs CLIPS of any user   */
/*   defined functions.  In the default case, there are no   */
/*   user defined functions.  To define functions, either    */
/*   this function must be replaced by a function with the   */
/*   same name within this file, or this function can be     */
/*   deleted from this file and included in another file.    */
/*   User defined functions may be included in this file or  */
/*   other files.                                            */
/*   Example of redefined usrfctns:                          */
/*     usrfuncs()                                            */
/*       {                                                   */
/*        define_function("fun1",'i',fun1);                  */
/*        define_function("other",'f',other);                */
/*       }                                                   */
/*************************************************************/
c_inv()
{
  char *body, *obs, *cbod, *cdat, *cpar;
  body = rstring(1);
  obs = rstring(2);
  cbod = rstring(3);
```

```
  cdat = rstring(4);
  cpar = rstring(5);
  inv2d_(body,obs,cbod,cdat,cpar);
  return(1);
}

c_creato()
{
 char *obs1;
 obs1 = rstring(1);
 creato_(obs1);
 return(1);
}

c_createm()
{
 char *mod;
 mod = rstring(1);
 creatm_(mod);
 return(1);
}

c_mlook()
{
 char *mod;
 mod=rstring(1);
 mlook_(mod);
 return(1);
}

c_slay()
{
 float lay, bloc;
 lay = rfloat(1);
 bloc = rfloat(2);
 slay_(&lay,&bloc);
 return(1);
}

c_sublay()
{
 float cn,cnb,cb,cl;
 cn = rfloat(1);
 cnb = rfloat(2);
 cb = rfloat(3);
 cl = rfloat(4);
 sublay_(&cn,&cnb,&cb,&cl);
}

c_fkread()
{
 char *name;
 name = rstring(1);
```

```
 fkread_(name);
}

c_fixpar()
{
 char *fob, *tp;
 float nbof, nbd;
 fob = rstring(1);
 tp = rstring(2);
 nbof = rfloat(3);
 nbd = rfloat(4);
 fixpar_(fob,tp,&nbof,&nbd);
}

c_modwt()
{
 float iw, w;
 iw = rfloat(1);
  w = rfloat(2);
 modwt_(&iw,&w);
}

c_fthrow()
{
  float fn,ft;
  fn = rfloat(1);
  ft = rfloat(2);
  fthrow_(&fn,&ft);
}

assert_(lst)
int lst;
{
  assert(lst);
  return(1);
}

escape_()
{
printf("\033");
}

literal_(alpha)
char *alpha;
{
printf("%c",*alpha);
}

psystem_()
{
 /* system("/users/ctsai/vplot/filters/tekpen outgf");*/
  system("/users/ctsai/pens/apen outgf");
}
```

```
usrfuncs()
  {
    extern mathfctns();
    extern float my_rand();
    mathfctns();
    define_function("rand",'f',my_rand,"my_rand");
    define_function("inv2d",'i',c_inv,"c_inv");
    define_function("creato",'i',c_creato,"c_creato");
    define_function("creatm",'i',c_creatm,"c_creatm");
    define_function("mlook",'i',c_mlook,"c_mlook");
    define_function("slay",'i',c_slay,"c_slay");
    define_function("sublay",'i',c_sublay,"c_sublay");
    define_function("fixpar",'i',c_fixpar,"c_fixpar");
    define_function("modwt",'i',c_modwt,"c_modwt");
    define_function("fthrow",'i',c_fthrow,"c_fthrow");
    define_function("fkread",'i',c_fkread,"c_fkread");
}
```

APPENDIX C

Listing of CLIPS Rule Base

```
(defrule enter-observed-data
  (initial-fact)
=>
(printout "Welcome to the Gravity Modeling Expert System" crlf)
(printout "Which way do you prefer to set up input message from :"crlf)
(printout "     1: file"crlf)
(printout "     2: terminal"crlf)
(printout "Your choice is (1 or 2) :")
(if (= 1 (read))
 then
  (printout "Enter the filename of the message : (without extension)"crlf)
  (bind ?fn (read))
  (assert (submit 1))
  (fkread ?fn)
 else
(printout "Your input message will be stored in the file \"mfile\""crlf)
(printout crlf"This expert system is to find the best possible model"crlf)
(printout "based on the given gravity anomalies and constraints."crlf)
(printout "Basically, there are two different data files needed."crlf)
(printout "One is the observed gravity data file, which includes"crlf)
(printout "X-coordinates, Z-coordinates and gravity anomalies of"crlf)
(printout "observation points; the other is the initial guess "crlf)
(printout "model file, which includes the number of faults, fault"crlf)
(printout "locations and fault dip angles, and the number of beds,"crlf)
(printout "bed dip angles and densities of beds etc."crlf)
(printout "You can enter the data from the terminal or from files."crlf)
(open "mfile" mfile "w")
(printout "Now, enter the observed data from :"crlf)
(printout "   1: file"crlf)
(printout "   2: terminal"crlf)
(printout "Your choice is (1 or 2) : ")
(bind ?f1 (read))
(assert (run 1))
(if (= ?f1 1)
 then
    (fprintout mfile "read observation from file"crlf)
    (printout crlf "NOTE: The data files must be named without extension"crlf)
    (printout crlf "What is the file name of the observed data ?"crlf)
    (bind ?fdata (read))
    (assert (gravity-data ?fdata))
 else
    (fprintout mfile "read observation from file"crlf)
    (printout "What is the file name to store the observed data ? "crlf)
    (bind ?fdata (read))
    (creato ?fdata)
    (assert (gravity-data ?fdata)))
(fprintout mfile "observation filename " ?fdata crlf)))

(defrule submit-observed-data
 (declare (salience 100))
 (submit 1)
 ?s1<-(read observation from ?f)
 ?s2<-(observation filename ?fdata)
```

```
=>
 (retract ?s1 ?s2)
 (if (eq ?f terminal)
  then
    (creato ?fdata))
 (assert (gravity-data ?fdata)))

(defrule submit-model-data
 (declare (salience 100))
 (submit 1)
 (gravity-data ?fdata)
 ?s1<-(read model from ?f2)
 ?s2<-(initial model filename ?fmod)
=>
 (retract ?s1 ?s2)
 (if (eq ?f2 terminal)
  then
     (creatm ?fmod))
 (assert (submit-layer))
 (mlook ?fmod)
 (assert (set-default 1))
 (assert (initial-model ?fmod)))


(defrule enter-model-data
  (not (submit ?xx))
  (gravity-data ?fdata)
=>
(printout "The initial model will be received from :"crlf)
(printout "     1: file"crlf)
(printout "     2: terminal"crlf)
(printout "Your choice is (1 or 2) : ")
(bind ?f2 (read))
(if (= ?f2 1)
 then
    (fprintout mfile "read model from file"crlf)
    (printout crlf"What is the file name of the initial model ?" crlf)
    (bind ?fmod (read))
    (assert (initial-model ?fmod))
    (mlook ?fmod)
    (assert (set-default 2))
 else
    (fprintout mfile "read model from file"crlf)
    (printout "What is the file name to store the initial model ?"crlf)
    (bind ?fmod (read))
    (creatm ?fmod)
    (assert (initial-model ?fmod))
    (mlook ?fmod)
    (assert (set-default 2)))
 (fprintout mfile "initial model filename " ?fmod crlf))

(defrule default-constraint
 (declare (salience 55))
```

```
 (depth ?dp)
 (width ?wd)
 (extent ?ext)
 (top ?top)
 ?s1<-(set-default ?xx)
=>
 (retract ?s1)
 (if (> ?xx 1)
  then
    (assert (set-layer)))
 (bind ?td (+ ?dp 0.0001))
 (bind ?lf (- 0 ?ext))
 (bind ?rt (+ ?wd ?ext))
 (assert (default-constraint b d -90. 90.))
 (assert (default-constraint b r 0. 3.5))
 (assert (default-constraint b z ?top ?td))
 (assert (default-constraint f d 0. 180.))
 (assert (default-constraint f x ?lf ?rt)))


(defrule set-layer
   (initial-model ?fmod)
   (set-layer)
=>
   (printout "Is your initial model layer-related ? (y/n) "crlf)
   (if (eq y (read))
      then
        (printout "How many layers are going to be considered ? "crlf)
        (bind ?l (read))
        (assert (write-layering))
        (assert (layer ?l))
        (assert (set-layer 1))
      else
        (assert (set-constraint))))


(defrule set1-layer
   (layer ?l)
   ?s <- (set-layer ?lay)
=>
   (retract ?s)
   (if (<= ?lay ?l)
      then
        (printout "How many different bodies in the layer #" ?lay " ?"crlf)
        (bind ?b (read))
        (slay ?lay ?b)
        (fprintout mfile "layers " ?lay " blocks " ?b crlf)
        (bind ?l1 (+ ?lay 1))
        (assert (set-layer ?l1))
      else
        (assert (set-constraint))))
```

```
(defrule set2-layer
 ?s1<-(lay ?n ?nb ?b ?l)
 ?s2<-(write-layering)
=>
 (retract ?s1 ?s2)
 (fprintout mfile "layering " ?n " number " ?nb " " ?b " " ?l crlf)
 (assert (write-layering)))

(defrule submit-layer
 (declare (salience 90))
 (layers ?n blocks ?blk)
 (submit 1)
 ?s1<-(layering ?n number ?nb ?b ?l)
 ?s2<-(submit-layer)
=>
 (retract ?s1 ?s2)
 (sublay ?n ?nb ?b ?l)
 (assert (submit-layer)))


(defrule set-constraint
  (gravity-data ?fdata)
  (initial-model ?mod)
  (set-constraint)
=>
  (printout "This program allows you set some constraints with" crlf)
  (printout "the final model calculated by an inversion program."crlf)
  (printout "You can set any value as the allowable maximum"crlf)
  (printout "difference between the initial and calculated model."crlf)
  (printout "For example, if you set a value 0.1 for one bed "crlf)
  (printout "as the density constraint, it means that the final"crlf)
  (printout "calculated density of that bed must not differ"crlf)
  (printout "from the initial value more than 0.1."crlf)
  (printout crlf"Do you have any constraint for the model ? (y/n) "crlf)
  (if (eq y (read))
   then
      (assert (set1-constraint))
   else
      (assert (set-weight))))


(defrule set1-constraint
?d1<-(set1-constraint)
=>
  (retract ?d1)
  (printout "Do you have any constraint for the faults ? (y/n) "crlf)
  (if (eq y (read))
   then
      (assert (fault-constraint))
   else
      (printout "Do you have any constraint for the beds ? (y/n) "crlf)
      (if (eq y (read))
       then
```

```
         (assert (bed-constraint))
      else
         (printout crlf crlf "No constraint for this model !! "crlf crlf)
         (assert (set-weight)))))

(defrule submit-fault-throw
 (declare (salience 85))
 ?s1<-(submit ?n&:(>= ?n 1))
 ?s2<-(fault ?f throw ?t)
=>
 (retract ?s1 ?s2)
 (fthrow ?f ?t)
 (assert (submit 2)))


(defrule set-throw
?d1<-(fault-constraint)
=>
 (retract ?d1)
 (printout "Do you want to set throws for the faults ? (y/n)"crlf)
 (if (eq y (read))
  then
    (printout "How many faults needed set throws for ? "crlf)
    (bind ?nf (read))
    (assert (no-fault ?nf))
    (assert (no1-fault 1))
    (assert (fault2-constraint))
  else
    (assert (fault2-constraint))))

(defrule set1-throw
  (declare (salience 5))
  (no-fault ?nf)
?s1<-(no1-fault ?n&:(<= ?n ?nf))
=>
  (retract ?s1)
  (printout "Which fault ? (ordered from left to right)"crlf)
  (bind ?x (read))
  (printout "Throw value ? (right up - ; right down +. unit Km)"crlf)
  (bind ?y (read))
  (fthrow ?x ?y)
  (fprintout mfile "fault " ?x " throw " ?y crlf)
  (bind ?n1 (+ 1 ?n))
  (assert (no1-fault ?n1)))


(defrule submit-fault-constraint
 (declare (salience 80))
 ?s1<-(submit ?n&:(>= ?n 1))
 ?s2<-(constraint f ?t ?fn ?fc)
=>
 (retract ?s1 ?s2)
 (bind ?fix (fixpar f ?t ?fn -1))
```

```
 (assert (constraint f ?t ?fn ?fc ?fix))
 (assert (unsatisfied-times f ?t ?fn 0))
 (assert (submit 2)))


(defrule set-fault-constraint
 ?s1 <- (fault2-constraint)
=>
  (retract ?s1)
  (printout "Do you want set constraints for fault parameters ? (y/n)"crlf)
  (if (eq y (read))
   then
  (printout "Which fault is to be constrained ? "crlf)
  (bind ?x1 (read))
  (printout "There are two parameters for each fault."crlf)
  (printout "Do you want set constraints for both parameters ? (y/n)"crlf)
  (if (eq y (read))
   then
     (printout "What is the allowable maximum deviation of the"crlf)
     (printout "dip angle for fault #" ?x1 ":"crlf)
     (bind ?n1 (read))
     (bind ?fix (fixpar f d ?x1 -1))
     (fprintout mfile "constraint f d " ?x1 " " ?n1 crlf)
     (assert (constraint f d ?x1 ?n1 ?fix))
     (printout "Enter the allowable maximum deviation of the "crlf)
     (printout "X-coordinate for fault #" ?x1 ":"crlf)
     (bind ?n2 (read))
     (bind ?fix (fixpar f x ?x1 -1))
     (fprintout mfile "constraint f x " ?x1 " " ?n2 crlf)
     (assert (constraint f x ?x1 ?n2 ?fix))
  else
     (printout "Which parameter do you want to constrain, dip or x ? (d/x) "crlf)
     (if (eq d (read))
      then
        (printout "What is the dip constraint value for fault #" ?x1 ":" crlf)
        (bind ?n3 (read))
        (bind ?fix (fixpar f d ?x1 -1))
        (fprintout mfile "constraint f d " ?x1 " " ?n3 crlf)
        (assert (constraint f d ?x1 ?n3 ?fix))
        (assert (unsatisfied-times f d ?x1 0))
      else
        (printout "What is the X-coordinate constraint value for fault #" ?x1 ":"crlf)
        (bind ?n3 (read))
        (bind ?fix (fixpar f x ?x1 -1))
        (fprintout mfile "constraint f x " ?x1 " " ?n3 crlf)
        (assert (unsatisfied-times f x ?x1 0))
        (assert (constraint f x ?x1  ?n3 ?fix))))
  (printout "Do you have more constraints for other faults ? (y/n) "crlf)
  (if (eq y (read))
   then
     (assert (fault-constraint))
   else
     (printout "Do you have any constraint for the beds ? (y/n) "crlf)
```

```
    (if (eq y (read))
     then
        (assert (bed-constraint))
     else
        (assert (set-weight))))
 else
   (printout "Do you have any constraint for the beds ? (y/n) "crlf)
   (if (eq y (read))
     then
       (assert (bed-constraint))
     else
       (assert (set-weight)))))

(defrule submit-beds-constraint
 (declare (salience 75))
 ?s1<-(submit ?n&:(>= ?n 1))
 ?s2<-(constraint b ?tp ?bk ?bd ?dt)
=>
 (retract ?s1 ?s2)
 (bind ?fix (fixpar b ?tp ?bk ?bd))
 (assert (unsatisfied-times b ?tp ?bk ?bd 0))
 (assert (constraint b ?tp ?bk ?bd ?dt ?fix))
 (assert (submit 2)))


(defrule set-beds-constraint
 ?s <- (bed-constraint)
=>
 (retract ?s)
 (printout "Which block ? (numbered from left to right) "crlf)
 (bind ?bk (read))
 (printout "Which bed ? (numbered from up to down) "crlf)
 (bind ?bd (read))
 (assert (block ?bk bed ?bd))
 (printout "Constraint for density ? (y/n)"crlf)
 (if (eq y (read))
  then
    (printout "Enter the allowable maximum deviation for density :"crlf)
    (bind ?dt (read))
    (bind ?fix (fixpar b r ?bk ?bd))
    (fprintout mfile "constraint b r " ?bk " " ?bd " " ?dt crlf)
    (assert (constraint b r ?bk ?bd ?dt ?fix))
    (assert (unsatisfied-times b r ?bk ?bd 0))
    (assert (bed1-constraint))
  else
    (assert (bed1-constraint))))


(defrule set1-beds-constraint
?s1 <- (bed1-constraint)
      (block ?bk1 bed ?bd1)
=>
 (retract ?s1)
```

```
 (printout "Constraint for depth Z ? (y/n)"crlf)
 (if (eq y (read))
  then
     (printout "Enter the allowable maximum deviation for depth :"crlf)
     (bind ?dz (read))
     (bind ?fix (fixpar b z ?bk1 ?bd1))
     (fprintout mfile "constraint b z " ?bk1 " " ?bd1 " " ?dz crlf)
     (assert (constraint b z ?bk1 ?bd1 ?dz ?fix))
     (assert (unsatisfied-times b z ?bk1 ?bd1 0))
     (assert (bed2-constraint))
  else
     (assert (bed2-constraint))))


(defrule set2-beds-constraint
?s2 <- (bed2-constraint)
?s3 <- (block ?bk1 bed ?bd1)
=>
 (retract ?s2 ?s3)
 (printout "Constraint for dip angle ? (y/n)" crlf)
 (if (eq y (read))
  then
     (printout "Enter the allowable maximum deviation for dip angle :"crlf)
     (bind ?dp (read))
     (bind ?fix (fixpar b d ?bk1 ?bd1))
     (fprintout mfile "constraint b d " ?bk1 " " ?bd1 " " ?dp crlf)
     (assert (constraint b d ?bk1 ?bd1 ?dp ?fix))
     (assert (unsatisfied-times b d ?bk1 ?bd1 0))
     (assert (bed3-constraint))
  else
     (assert (bed3-constraint))))

(defrule set3-beds-constraint
?s3 <- (bed3-constraint)
=>
 (retract ?s3)
 (printout "More constraint for other beds ? (y/n) "crlf)
 (if (eq y (read))
  then
     (assert (bed-constraint))
  else
     (assert (set-weight))))



(defrule set-weight
  (not (submit ?xx))
  (set-weight)
  ?s<-(run ?tm&:(< ?tm 2))
=>
  (retract ?s)
  (printout "Enter the filename of calculated data :" crlf)
  (bind ?cdata (read))
```

```
  (fprintout mfile "calculated data filename " ?cdata crlf)
  (assert (calculated-data ?cdata))
  (printout "Enter RMS error limit :" crlf)
  (bind ?rms (read))
  (assert (submit 2))
  (fprintout mfile "rmse limit " ?rms crlf)
  (assert (rmslimit ?rms)))


(defrule submit-weight
  (declare (salience 50))
  (calculated-data ?cf)
  (rmslimit ?rms)
  ?s1<-(submit ?xx)
=>
  (open "parfl" parfl "w")
  (bind ?dp1 0.1)
  (bind ?af1 3)
  (bind ?dp2 0.5)
  (bind ?af2 1)
  (bind ?dp3 0.1)
  (bind ?af3 3.5)
  (bind ?dp4 0.5)
  (bind ?af4 1)
  (bind ?dp5 0.01)
  (bind ?af5 0.3)
  (assert (increment f x ?dp1))
  (assert (weighting f x ?af1))
  (assert (increment f d ?dp2))
  (assert (weighting f d ?af2))
  (assert (increment b z ?dp3))
  (assert (weighting b z ?af3))
  (assert (increment b d ?dp4))
  (assert (weighting b d ?af4))
  (assert (increment b r ?dp5))
  (assert (weighting b r ?af5))
  (format parfl "%5d %n" 1)
  (format parfl "%12.5f %n " ?rms)
  (format parfl "%12.5f %12.5f %n %12.5f %12.5f %n" ?dp1 ?af1 ?dp2 ?af2)
  (format parfl "%12.5f %12.5f %n %12.5f %12.5f %n" ?dp3 ?af3 ?dp4 ?af4)
  (format parfl "%12.5f %12.5f %n" ?dp5 ?af5)
  (assert (run 2))
  (assert (set1-weight)))

(defrule submit1-weight
  (submit ?xx)
?s1<-(calculated data filename ?cf)
?s2<-(rmse limit ?rms)
=>
  (retract ?s1 ?s2)
  (assert (calculated-data ?cf))
  (assert (rmslimit ?rms)))
```

```
(defrule run1-inversion
  (declare (salience -15))
  (gravity-data ?fdata)
  (initial-model ?fmod)
  (not (calculated-model ?nmod))
  (calculated-data ?cdata)
  ?s1<-(set1-weight)
=>
  (retract ?s1)
  (assert (parameters parfl))
  (close parfl)
  (inv2d ?fmod ?fdata sumod ?cdata parfl)
  (assert (check-results)))


(defrule set1-weight
 (declare (salience -5))
 ?s2<-(constraint f ?tp ?n ?v ?fix)
 (increment f ?tp ?p&:(< ?v ?p))
 ?s1<-(set1-weight)
=>
 (retract ?s1 ?s2)
 (bind ?fix (fixpar f ?tp ?n -1 ))
 (format parfl "%5d %n" ?fix)
 (assert (used constraint f ?tp ?n ?v ?fix))
 (assert (set1-weight)))


(defrule set2-weight
 (declare (salience -10))
 ?s2<-(constraint b ?tp ?bk ?bd ?v ?fix)
 (increment b ?tp ?p&:(< ?v ?p))
 ?s1<-(set1-weight)
=>
 (retract ?s1 ?s2)
 (bind ?fix (fixpar b ?tp ?bk ?bd ))
 (format parfl "%5d %n" ?fix)
 (assert (used constraint b ?tp ?bk ?bd ?v ?fix))
 (assert (set1-weight)))


(defrule check-fault
 (declare (salience -10))
 ?s1 <- (check-results)
 ?s4 <- ( $? constraint f ?t ?n ?v ?fix)
 ?s5 <- (unsatisfied-times f ?t ?n ?tms)
 ?s2 <- (result f ?t ?n ?m ?c ?fix)
 ?s3 <- (weight ?fix ?w)
=>
 (retract ?s1 ?s2)
 (bind ?df (abs (- ?m ?c)))
 (if (<= ?df ?v)
  then
```

```
     (assert (new f ?t ?n ?c))
  else
     (retract ?s5)
     (bind ?tms1 (+ ?tms 1))
     (assert (unsatisfied-times f ?t ?n ?tms1))
     (if (>= ?df (* 5 ?v))
       then
         (retract ?s4)
         (if (> ?c ?m)
           then
             (bind ?nm (+ ?m ?v))
           else
             (bind ?nm (- ?m ?v)))
         (assert (new f ?t ?n ?nm))
         (assert (constraint f ?t ?n 0 ?fix))
       else
         (retract ?s3)
         (bind ?nw (/ ?w 2))
         (assert (new f ?t ?n ?m))
         (bind ?fix (fixpar f ?t ?n -1))
         (modwt ?fix ?nw)
         (assert (weight ?fix ?nw)))
     (assert (write-model))
     (assert (bad f ?t ?n)))
 (assert (check-results)))

(defrule check-layer
 (declare (salience -20))
 ?s1 <- (check-results)
 ?s4 <- ( $? constraint b ?t ?b ?l ?v ?fix)
 ?s5 <- (unsatisfied-times b ?t ?b ?l ?tms)
 ?s2 <- (result b ?t ?b ?l ?m ?c ?fix)
 ?s3 <- (weight ?fix ?w)
=>
 (retract ?s1 ?s2)
 (bind ?df (abs (- ?m ?c)))
 (if (<= ?df ?v)
  then
     (assert (new b ?t ?b ?l ?c))
  else
     (retract ?s5)
     (bind ?tms1 (+ ?tms 1))
     (assert (unsatisfied-times b ?t ?b ?l ?tms1))
     (if (>= ?df (* 5 ?v))
       then
         (retract ?s4)
         (if (> ?c ?m)
           then
             (bind ?nm (+ ?m ?v))
           else
             (bind ?nm (- ?m ?v)))
         (assert (new b ?t ?b ?l ?nm))
         (assert (constraint b ?t ?b ?l 0 ?fix))
```

```
        else
          (retract ?s3)
          (bind ?nw (/ ?w 2))
          (bind ?fix (fixpar b ?t ?b ?1))
          (modwt ?fix ?nw)
          (assert (weight ?fix ?nw))
          (assert (new b ?t ?b ?1 ?m)))
      (assert (write-model))
      (assert (bad b ?t ?b ?1)))
 (assert (check-results)))

(defrule write-new-model
     (declare (salience -50))
     (depth ?x1)
     (width ?x2)
     (extent ?x3)
     (top ?x4)
     (num-fault ?x5)
     (mean-density ?x6)
     ?d1<-(write-model)
=>
  (retract ?d1)
  (open "nmod" nmod "w")
  (format nmod "%7.2f %7.2f %7.2f %7.2f %3d %7.2f %n" ?x1 ?x2 ?x3 ?x4 ?x5 ?x6)
  (assert (write-fault))
  (assert (fault 1)))

(defrule write1-new-model
     (num-fault ?x1)
?d1<-(write-fault)
?d2<-(fault ?x2)
?d3<-(new f x ?x2 ?v1)
?d4<-(new f d ?x2 ?v2)
=>
  (retract ?d2 ?d3 ?d4)
  (format nmod "%7.2f %7.2f %n" ?v1 ?v2)
  (bind ?x3 (+ ?x2 1))
  (if (= ?x2 ?x1)
   then
     (retract ?d1)
     (assert (write-layer))
     (assert (section 1))
   else
     (assert (fault ?x3))))


(defrule write3-new-model
     (write-layer)
 ?s1<-(section ?xb)
     (section ?xb ?xx ?x1)
=>
  (retract ?s1)
  (format nmod "%7.2f %3d %n" ?xx ?x1)
```

```
  (assert (layer ?xb ?xl 1)))

(defrule write4-new-model
  (declare (salience -10))
?d1<-(layer ?xb ?xl ?x)
?d2<-(new b z ?xb ?x ?v1)
?d3<-(new b d ?xb ?x ?v2)
?d4<-(new b r ?xb ?x ?v3)
=>
 (retract ?d1 ?d2 ?d3 ?d4)
 (format nmod "%7.2f %7.2f %7.2f %n" ?v1 ?v2 ?v3)
 (if (= ?xl ?x)
  then
     (bind ?xb1 (+ ?xb 1))
     (assert (section ?xb1))
  else
     (bind ?xl1 (+ ?x 1))
     (assert (layer ?xb ?xl ?xl1))))


(defrule write5-new-model
 (declare (salience -52))
?d1<-(write-layer)
?d2<-(section ?x)
=>
 (retract ?d1 ?d2)
 (close nmod)
 (assert (calculated-model nmod))
 (assert (set2-weight)))


(defrule write-new-pars
  ?s<-(run ?tm&:(> ?tm 1))
  ?s1<-(set2-weight)
=>
  (retract ?s ?s1)
  (bind ?tm1 (+ ?tm 1))
  (open "parfl" parfl "w")
  (format parfl "%5d %n" ?tm1)
  (assert (run ?tm1))
  (assert (set1-weight)))




(defrule delete-bad
 (declare (salience 10))
?d1<-(bad $?x)
?d2<-(set1-weight)
=>
 (retract ?d1 ?d2)
 (assert (rerun inversion))
```

```
 (assert (set1-weight)))

(defrule delete-weight
  (declare (salience 10))
  ?s1<-(weight $?x)
  ?s2<-(set1-weight)
=>
  (retract ?s1 ?s2)
  (assert (set1-weight)))


(defrule write-no-constraint
 (declare (salience -30))
?d1<-(check-results)
?d2<-(result f ?t ?n ?m ?c ?fix)
 (weight ?fix ?w)
 (default-constraint f ?t ?low ?hi)
=>
 (retract ?d1 ?d2)
 (assert (write-model))
 (if (eq ?t d)
  then
     (bind ?t1 dip)
  else
     (bind ?t1 location))
 (bind ?dc (abs (- ?m ?c)))
 (if (< ?c ?low)
  then
    (if (< ?m ?low)
     then
       (printout "Check model parameter " ?n " fault " ?t1 crlf)
       (printout "It should be within " ?low " and " ?hi crlf)
       (halt)
     else
       (bind ?nw (/ ?w 10))
       (bind ?cc (+ ?m (/ ?dc 100))))
  else
    (if (> ?c ?hi)
     then
       (if (> ?m ?hi)
        then
          (printout "Check model parameter " ?n " fault " ?t1 crlf)
          (printout "It should be within " ?low " and " ?hi crlf)
          (halt)
        else
          (bind ?nw (/ ?w 10))
          (bind ?cc (- ?m (/ ?dc 100))))
      else
        (if (< ?dc (/ ?m 10000))
         then
           (bind ?nw (* ?w 3))
           (bind ?cc (+ ?m (* (- ?c ?m) 5)))
         else
```

```
            (if (< ?dc (/ ?m 1000))
             then
               (bind ?rm (rand))
               (bind ?rm1 (mod ?rm 10))
               (bind ?cc (/ (+ ?m (* 2 ?c)) 3))
               (if (< ?rm1 5)
                then
                  (if (= ?c 0.)
                   then
                     (bind ?nw 1.2)
                   else
                     (bind ?nw (* ?w (/ (* ?m ?m) (* ?c ?c)))))
                else
                  (if (= ?m 0.)
                   then
                     (bind ?nw 0.8)
                   else
                     (bind ?nw (* ?w (/ (* ?c ?c) (* ?m ?m))))))
             else
               (bind ?nw (/ ?w 3))
               (bind ?cc ?c)))))
 (assert (new f ?t ?m ?cc))
 (modwt ?fix ?nw)
 (assert (check-results)))

(defrule write1-no-constraint
 (declare (salience -40))
?d1<-(check-results)
?d2<-(result b ?t ?b ?l ?m ?c ?fix)
 (weight ?fix ?w)
 (default-constraint b ?t ?low ?hi)
=>
 (retract ?d1 ?d2)
 (assert (write-model))
 (bind ?dc (abs (- ?m ?c)))
 (if (eq ?t d)
  then
     (bind ?t1 dip)
  else
     (if (eq ?t r)
      then
         (bind ?t1 density)
      else
         (bind ?t1 depth)))
 (if (< ?c ?low)
  then
    (if (< ?m ?low)
     then
       (printout "Check model parameter block " ?b " layer " ?l " " ?t1 crlf)
       (printout "It should be within " ?low " and " ?hi crlf)
       (halt)
     else
       (bind ?nw (/ ?w 10))
```

```
       (bind ?cc (+ ?m (/ ?dc 100))))
  else
    (if (> ?c ?hi)
     then
       (if (> ?m ?hi)
        then
          (printout "Check model parameter block " ?b " layer " ?l " " ?tl crlf)
          (printout "It should be within " ?low " and " ?hi crlf)
          (halt)
        else
          (bind ?nw (/ ?w 10))
          (bind ?cc (+ ?m (/ ?dc 100))))
      else
        (if (< ?dc (/ ?m 10000))
         then
           (bind ?nw (* ?w 3))
           (bind ?cc (- ?m (* ?dc 2)))
         else
           (if (< ?dc (/ ?m 1000))
            then
              (bind ?rm (rand))
              (bind ?rm1 (mod ?rm 10))
              (bind ?cc (+ ?m (* ?dc 2)))
              (if (< ?rm1 5)
               then
                 (if (= ?c 0.)
                  then
                    (bind ?nw 1.2)
                  else
                    (bind ?nw (* ?w (/ (* ?m ?m) (* ?c ?c)))))
               else
                 (if (= ?m 0.)
                  then
                    (bind ?nw 0.8)
                  else
                    (bind ?nw (* ?w (/ (* ?c ?c) (* ?m ?m))))))
             else
               (bind ?nw (/ ?w 3))
               (bind ?cc ?c)))))
 (modwt ?fix ?nw)
 (assert (new b ?t ?b ?l ?cc))
 (assert (check-results)))

(defrule run2-inversion
 (declare (salience -60))
 (gravity-data ?fdata)
 (initial-model ?fmod)
 (calculated-model ?nmod)
 (calculated-data ?cdata)
?d1<-(set1-weight)
?d2<-(check-results)
?d3<-(rerun inversion)
?d4<-(calculated-rmse ?rms)
```

```
=>
 (retract ?d1 ?d2 ?d3 ?d4)
 (close parfl)
 (inv2d ?fmod ?fdata ?nmod ?cdata parfl)
 (assert (check-results)))

(defrule run3-inversion
 (declare (salience -55))
 (rmslimit ?rm1)
 (calculated-rmse ?rm2&:(> ?rm2 ?rm1))
=>
 (assert (set1-weight))
 (assert (rerun inversion)))

(defrule release-bed-constraints
 (declare (salience -53))
 ?s1 <- (unsatisfied-times b ?t ?b ?l ?tms&:(> ?tms 10))
 ?s2 <- ($? constraint b ?t ?b ?l ?v ?fix)
=>
 (retract ?s1)
 (if (eq r ?t)
  then
    (bind ?type density)
  else
    (if (eq d ?t)
     then
       (bind ?type dip)
     else
       (bind ?type depth)))
 (printout "The bed " ?type " constraint of block " ?b " layer " ?l " is hard to be
  satisfied with" crlf)
 (printout "Would you change it ? (y/n)"crlf)
 (if (eq y (read))
  then
    (retract ?s2)
    (printout "The old maximum allowable range is " ?v crlf)
    (printout "What is the new maximum allowable range ? ( >100 : delete it)"crlf)
    (bind ?nv (read))
    (if (< ?nv 100)
     then
       (assert (constraint b ?t ?b ?l ?nv ?fix))))
 (assert (unsatisfied-times b ?t ?b ?l 0))
 (assert (set1-weight))
 (assert (rerun-inversion)))
(defrule release-fault-constraint
 (declare (salience -53))
?s1 <- (unsatisfied-times f ?t ?n ?tms&:(> ?tms 10))
?s2 <- ($? constraint f ?t ?n ?v ?fix)
=>
 (retract ?s1)
 (if (eq x ?t)
  then
    (bind ?type x-location)
```

```
 else
   (bind ?type dip))
(printout "The " ?type " constraint of " ?n " fault is hard to be satisfied with " crlf)
(printout "Would you change it ? (y/n)"crlf)
(if (eq y (read))
 then
   (retract ?s2)
   (printout "The old maximum allowable range is " ?v crlf)
   (printout "What is the new maximum allowable range ? ( >100 : delete it)"crlf)
   (bind ?nv (read))
   (if (< ?nv 100)
    then
      (assert (constraint f ?t ?n ?nv ?fix))))
(assert (unsatisfied-times f ?t ?n 0))
(assert (set1-weight))
(assert (rerun-inversion)))
```